**CS161: Design and Analysis of Algorithms** **Summer 2004**

# Problem Set #2 Solutions

## General Notes

- **Regrade Policy:** If you believe an error has been made in the grading of your problem set, you may resubmit it for a regrade. If the error consists of more than an error in addition of points, please include with your problem set a detailed explanation of which problems you think you deserve more points on and why. We reserve the right to regrade your entire problem set, so your final grade may either increase or decrease.
- Remember you can only split an expectation of a product into the product of the expectations if the two terms are **independent**. Many students did this on question 3c either without justification or justifying it as linearity of expectation, which is incorrect.
- For the skip list question 4b, many students found the expected number of nodes between $left[i+1].below$ and $right[i+1].below$ by saying the expected number of nodes at level $i+1$ is $np^{i+1}$ and the expected number of nodes at level $i$ is $np^i$. Then, if you divide the number of nodes at $i$ by the number at $i+1$, you get an average interval of $1/p$. While this gives the correct answer, it is not a valid argument as it stands. For example, let's say you always propagate the first nodes in level $i$ - you will get an interval of length 1 with probability $1/p$ and an interval of length $np^i(1-p)$ with probability $1-1/p$, which clearly gives you a different expectation. You need to use properties other than just the expected number of nodes to calculate the expected interval length.

1. [**18 points**] Probability and Conditional Probability

   Throughout this problem, please justify your answers mathematically as well as logically; however, you do not have to give any formal proofs.

   (a) [**5 points**] Do problem C.2-9 on page 1106 of CLRS. Assume that initially, the prize is placed behind one of the three curtains randomly.

   **Answer:** The answer is not 1/2, because now the two curtains are no longer equally likely to hide the prize. In fact, the original 3 cases are still equally likely, but now in two of the cases the prize is behind the curtain that is not revealed or chosen. Therefore, the probability of winning if you stay is 1/3, if you switch is 2/3.

   (b) [**5 points**] Do problem C.2-10 on page 1106 of CLRS.

   **Answer:** This is essentially the same problem, except now there is no option to switch. The prisoners are the curtains, freedom is the prize, and the guard's revelation is equivalent to the emcee's revelation in the previous question. Thus, the probability that $X$ will go free is 1/3.

(c) [**8 points**] Consider the following game using a standard 52-card deck (with 26 red cards and 26 black cards):

i. the deck $D$ is shuffled randomly.
ii. one at a time, the top card of $D$ is removed and revealed until you say "stop" or only one card remains.
iii. if the next card is red, you win the game. if the next card is black, you lose the game.

Consider the following algorithmic strategy for playing this game:

i. count the number of black cards (b) and red cards (r) revealed thus far.
ii. as soon as $b > r$, say "stop".

What is the probability of success for this strategy? Is there a better strategy? Can you prove a useful upper bound on the probability of success for any legal strategy?

**Hint:** Consider the following modification to the game: after you say "stop", rather than looking at the top card of $D$, we look at the bottom card.

**Answer:** For any given strategy $S$, let $Pr_S\{(b,r)\}$ be the probability that we stop after we have seen $b$ black cards and $r$ red cards. Note that some of these may be 0, for example, using the strategy in question we will only stop when $b = r + 1$ or $(b,r) = (25,26)$.

Since we assume all permutations of the deck to be equally likely, the remaining $52 - b - r$ cards are in random order, so the probability of success (top remaining card is red) is just

$$Pr_S\{\text{success} \mid (b,r)\} = \frac{26-r}{52-b-r}.$$

Thus, we can write

$$Pr_S\{\text{success}\} = \sum_{(b,r)} Pr_S\{(b,r)\} \cdot \frac{26-r}{52-b-r}.$$

Now notice that $(26-r)/(52-b-r)$ is also the probability that the bottom card is red given that we stop at $(b,r)$. Thus, we can say

$$Pr_S\{\text{success}\} = \sum_{(b,r)} Pr_S\{(b,r)\} \cdot Pr\{\text{bottom} \;=\; \text{red} \mid (b,r)\}.$$

Now we manipulate the sum by multiplying the two probabilities:

$$Pr_S\{\text{success}\} = \sum_{(b,r)} Pr\{\text{bottom} \;=\; \text{red} \;\wedge (b,r)\} = Pr\{\text{bottom} \;=\; \text{red}\} = 1/2.$$

The argument applies for any legal strategy, thus, regardless of the strategy, we succeed with probability 1/2.

If you are uncomfortable with some of the probabilistic manipulation and reasoning in this solution, please review Appendix C in CLRS, particularly section $C.2$ and Bayes's theorem.

2. [**12 points**] Modifying Quicksort

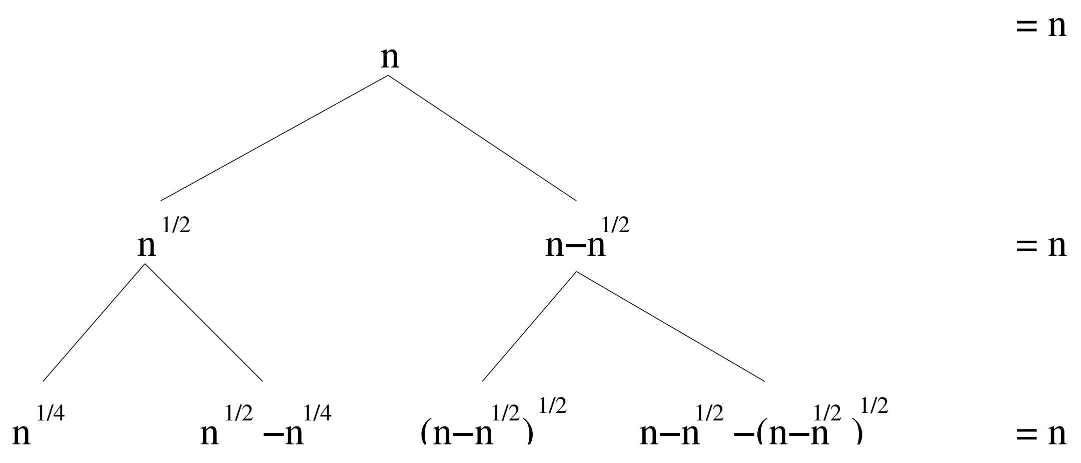
   Consider the following modification to the deterministic quicksort algorithm: instead of using $A[n]$ as pivot, take the last $2\sqrt{n}+1$ elements of $A$, sort them using insertion sort, then use the middle element of these as the pivot. Assume that this guarantees there are at least $\sqrt{n}$ elements in each subarray after partitioning - we will ignore the issue of repeated elements in this problem. As usual, ignore the issue of rounding as well.

   (a) [**3 points**] Write a recurrence for the worst-case running time $T(n)$ of this algorithm.

   **Answer:** Our partition guarantees that there are at least $\sqrt{n}$ elements in each subarray. In the worst case, all the remaining elements will go to one subarray. Then, we will need to solve one subproblem of the size $\sqrt{n}$ and one of the size $n-\sqrt{n}$. Also, we need to sort the $2\sqrt{n}+1$ elements to find the pivot. Using insertion sort, this is a worst case running time of $\Theta((2\sqrt{n}+1)^2) = \Theta(n)$. Then, we find the median of the sorted sample, a $\Theta(1)$ step. In addition, we need to partition the elements at each step, a $\Theta(n)$ operation. So in the worst-case, the recurrence relation will be $T(n) = T(\sqrt{n}) + T(n-\sqrt{n}) + \Theta(n)$.

   (b) [**2 points**] Draw the recursion tree for the expression you came up with in part (a). Label the work in the top 3 levels of the tree.

   **Answer:**



   (c) [**6 points**] How many levels does it take before the shortest branch reaches a leaf? the longest branch (an asymptotic upper bound is sufficient)?

   **Hint:** Consider writing a recurrence for the height of the longest branch. You may find the approximation $\sqrt{n-\sqrt{n}} \approx \sqrt{n} - 1/2$ helpful in guessing a solution for this recurrence.

   **Answer:** The shortest branch is the one where everytime we call $T(\sqrt{n})$. At each level $i$, we will call $T(n^{1/2^i})$. We assume $T(n)$ is a constant for small values of $n$, so we find the value of $i$ for which $n^{1/2^i} \leq 2$. This is true for $i \geq \lg\lg n$. Thus,

the shortest branch reaches a leaf in $\lg\lg n$ steps.
The longest branch is the one where everytime we call $T(n-\sqrt{n})$. At the second level we will call $T(n-\sqrt{n}-\sqrt{n-\sqrt{n}}) \approx T(n-\sqrt{n}-\sqrt{n}-1/2)$. So, at this level, we essentially subtract another $\sqrt{n}$. Therefore, we can guess that at each level $i$, we call $T(n-i\sqrt{n})$ which will be a constant when $i=\sqrt{n}$. We can now check to see that this is in fact an upper bound for the number of levels. Define $h(n)$ to be the number of levels in the recurrence tree. We know that $h(n) = h(n-\sqrt{n})+1$, since we add one level each time we recurse. We guess that $h(i) \leq c\sqrt{i}$ for $i < n$ and check.

$$\begin{aligned} h(n) &= h(n-\sqrt{n})+1 \\ &\leq c\sqrt{n-\sqrt{n}}+1 \\ &\approx c(\sqrt{n}-1/2)+1 \\ &= c\sqrt{n}-c/2+1 \\ &\leq c\sqrt{n} \end{aligned}$$

which is true for $c \geq 2$. Therefore, we know that the longest branch is $O(\sqrt{n})$.

(d) **[1 point]** Using your answers from parts (b) and (c), give a big-O bound on $T(n)$.
**Answer:** $T(n) = O(n^{3/2})$. Looking at the recurrence tree from part (b), we see that we do $\Theta(n)$ work on each level. We showed in part (c) that the longest branch of the recurrence tree had a height of $O(\sqrt{n})$ levels. So, we know the recurrence is bounded by $T(n) = O(n^{3/2})$.

3. **[15 points]** Alternative Quicksort Analysis

In this problem, you will analyze the running time of the randomized quicksort algorithm using a different approach than in lecture. Instead of counting the expected number of comparisons, this approach focuses on the expected running time of each recursive call to randomized quicksort.

Throughout your analysis, please be as formal as possible.

(a) **[2 points]** Do problem 7-2(a) on page 160 of CLRS.
**Answer:** We know that we select pivots from the array uniformly at random. Since there are $n$ elements in the array, and since the sum of the probabilities of choosing any one element as the pivot must equal one, each element must be chosen with probability $1/n$. If we define the indicator random variable $X_i = I\{i^{th}$ smallest element chosen as the pivot$\}$, then $E[X_i]$ is simply $\sum_{j=1}^{n} Pr\{\text{select } j \wedge j$ is the $i^{th}$ smallest element$\}$. These are independent, so we can break apart the product and get $\sum_{j=1}^{n} Pr\{\text{select } j\} \cdot Pr\{j$ is the $i^{th}$ smallest element$\} = \sum_{j=1}^{n}(1/n)\cdot(1/n)$, since each element is equally likely to be the $i^{th}$ smallest element. This gives $E[X_i] = 1/n$.

(b) [**3 points**] Do problem 7-2(b) on page 161 of CLRS.

**Answer:** The expected running time of quicksort depends on the sizes of the subarrays after partitioning. We can sum over the sizes resulting from all possible pivots multiplied by the probability of that pivot. If the pivot is at $q$, then the resulting subarrays will have size $q-1$ and $n-q$. In addition, it takes $\Theta(n)$ time to do the partitioning. So, the total expected running time is:

$E[T(n)] = E[\sum_{q=1}^{n} X_q(T(q-1) + T(n-q) + \Theta(n))]$

(c) [**2 points**] Do problem 7-2(c) on page 161 of CLRS.

**Answer:** By linearity of expectation, we can move the summation outside of the expectation.

$$E[T(n)] = \sum_{q=1}^{n} E[X_q(T(q-1) + T(n-q) + \Theta(n))]$$

Because $X_q$ is independent from the time is takes to do the recursive calls, we can split the expectation of the product into the product of expectations.

$$E[T(n)] = \sum_{q=1}^{n} E[X_q] \cdot E[T(q-1) + T(n-q) + \Theta(n)]$$

Substituting in the value for $E[X_q]$ that we solved in part (a) and again changing the expectation of the sum into a sum of expectations, we have

$$\begin{aligned} E[T(n)] &= \sum_{q=1}^{n} \frac{1}{n}(E[T(q-1)] + E[T(n-q)] + E[\Theta(n)]) \\ &= \frac{1}{n}\sum_{q=1}^{n}(E[T(q-1)] + E[T(n-q)] + \Theta(n)) \\ &= \frac{1}{n}(\sum_{q=1}^{n} E[T(q-1)] + \sum_{q=1}^{n} E[T(n-q)] + \sum_{q=1}^{n} \Theta(n)) \\ &= \frac{1}{n}((E[T(0)] + E[T(1)] + \cdots + E[T(n-1)]) + \\ &\quad (E[T(n-1)] + E[T(n-2)] + \cdots + E[T(0)]) + n\Theta(n)) \\ &= \frac{1}{n}(2\sum_{q=2}^{n-1} E[T(q)]) + \Theta(n) \end{aligned}$$

since we consider $E[T(0)]$ and $E[T(1)]$ to be $\Theta(1)$.

(d) [**6 points**] Do problem 7-2(d) on page 161 of CLRS.

**Answer:**

$$\begin{aligned}
\sum_{k=2}^{n-1} k \lg k &= \sum_{k=2}^{\lceil n/2\rceil -1} k \lg k + \sum_{k=\lceil n/2\rceil}^{n-1} k \lg k \\
&\leq \sum_{k=2}^{\lceil n/2\rceil -1} k \lg(n/2) + \sum_{k=\lceil n/2\rceil}^{n-1} k \lg n \\
&= \lg(n/2) \sum_{k=2}^{\lceil n/2\rceil -1} k + \lg n \sum_{k=\lceil n/2\rceil}^{n-1} k \\
&= \lg(n/2)\left(\frac{(n/2-1)(n/2)}{2} - 1\right) + \lg(n)\left(\frac{(n-1)(n)}{2} - \frac{(n/2-1)(n/2)}{2}\right) \\
&\leq (\lg n - \lg 2)\frac{(n/2)^2}{2} + (\lg n)\left(\frac{n^2}{2} - \frac{(n/2)^2}{2}\right) \\
&= (\lg n)\frac{n^2}{8} - \frac{n^2}{8} + (\lg n)\frac{n^2}{2} - (\lg n)\frac{n^2}{8} \\
&= \frac{1}{2}n^2 \lg n - \frac{1}{8}n^2
\end{aligned}$$

(e) **[2 points]** Do problem 7-2(e) on page 161 of CLRS. Note that you are actually showing $E[T(n)] = O(n \lg n)$, but the $\Omega$ bound is trivial.

**Answer:** We will show this using the guess-and-check method. First, we guess that $E[T(q)] \leq cq \lg q$ for $2 \leq q < n$. Then, we show that the same is true for $E[T(n)]$.

$$\begin{aligned}
E[T(n)] &= \frac{2}{n}\sum_{q=2}^{n-1} E[T(q)] + kn \\
&\leq \frac{2}{n}\sum_{q=2}^{n-1} cq \lg q + kn \\
&\leq \frac{2}{n}c\left(\frac{1}{2}n^2 \lg n - \frac{1}{8}n^2\right) + kn \\
&= 2c\left(\frac{1}{2}n \lg n - \frac{1}{8}n\right) + kn \\
&= cn \lg n - \frac{1}{4}cn + kn \\
&\leq cn \lg n
\end{aligned}$$

for $c \geq 4k$.

4. **[19 points]** Skip List Analysis

In this problem, you will analyze the behavior of the skip list data structure introduced in lecture. Please see the Skip List Summary handout for more information, including detailed descriptions of the FIND, INSERT, and DELETE operations.

Throughout this problem, we will consider a skip list with $L$ levels and $n$ elements, in which elements are propagated to the next level with probability $p$, $0 < p < 1$.

(a) **[5 points]** What is the expected value of $L$ in terms of $n$ and $p$? For simplicity of analysis, consider levels that you expect to contain less than 1 cell to be nonexistent. In practice, how might you enforce this assumption?

**Answer:** $E[L] = \Theta(\log_{1/p} n)$. We can define the indicator variable $I_{i,k}$ to be equal to 1 if and only if the element $A[i]$ is present in the $k^{th}$ level of the skip list. Then,

$$E[L] = \min_k (E[\sum_{j=1}^{n} I_{j,k}] \leq 1)$$

the expected number of levels in the skip list is equal to the minimum level where the expected number of elements in the list is $\leq 1$. By linearity of expectation, we can move the summation outside of the expectation:

$$E[L] = \min_k (\sum_{j=1}^{n} E[I_{j,k}] \leq 1)$$

The expectation of an indicator variable is simply the probability of that event. What is the probability that element $A[i]$ is present in the $k^{th}$ level of the skip list? It is the probability that we propagated $A[i]$ up from level 0 to the level $k$. This is the product of $k$ independent propagation events, each of which has probability $p$. Thus, the equation for the expected number of levels is

$$\begin{aligned}
E[L] &= \min_k (\sum_{j=1}^{n} p^k \leq 1) \\
&= \min_k (p^k \sum_{j=1}^{n} 1 \leq 1) \\
&= \min_k (np^k \leq 1)
\end{aligned}$$

The value $np^k$ falls below 1 when $k \geq \log_{1/p} n$. So, we expect to have $E[L] = \Theta(\log_{1/p} n)$.

(b) **[8 points]** Consider the work done by the SEARCH operation at level $i$, where $i < L$. We must examine, in the worst case, all cells between $left[i+1].below$ and $right[i+1].below$ (since $left[L]$ and $right[L]$ are not actually defined, we can say that in level $L-1$ we look between *dummy* and NULL). Thus, we examine all cells immediately after some given cell in level $i$ that have not been propagated to level $i+1$. What is the expected number of such cells? What is the expected running time of the SEARCH operation in terms of $n$ and $p$?

**Answer:** We are trying to find the expected number of elements examined on level $i$, which is equal to the number of elements between $left[i+1].below$ and $right[i+1].below$. This is equal to the expected number of elements in level $i$ between two elements which are propagated up to level $i+1$. Since we know at least one element is propagated (the *dummy* element is always propagated), we simply compute the expected number of elements until another element is propagated. Since each element is propagated independently with probability $p$, the probability that the $k$th next element is the first to be propagated is $(1-p)^{k-1}p$. Therefore, the expected value of $k$ is $E[k] = \sum_{k=1}^{\infty} kp(1-p)^{k-1}$. This is the expectation of the geometric distribution, so the expectation is given by $E[k] = 1/p$ (p. 1112 CLRS). So, we expect to have to search through $1/p$ elements at each of the $i$ levels. Since we found in part (a) that there are $\Theta(\log_{1/p} n)$ levels, we expect SEARCH to take time $\Theta((1/p)\log_{1/p} n) = \Theta((1/p)\log_{1/p} n)$.

(c) **[3 points]** Use your answers to the previous parts to find the expected running time of the INSERT and DELETE operations.

**Answer:** INSERT and DELETE first call SEARCH, which runs in expected time $\Theta((1/p)lg_{1/p}n)$. Inserting and deleting an element from each level of the skip list takes $\Theta(1)$ time. The expected number of levels that each element is propagated is $E[levels] = \sum_{k=1}^{\infty} kp^{k-1}(1-p) = 1/(1-p)$, since this is once again the expectation of a geometric distribution. So, the total expected running time for INSERT and DELETE is $\Theta((1/p)\lg_{1/p} n) + (1/(1-p))\Theta(1) = \Theta((1/p)\lg_{1/p} n + 1/(1-p))$. Notice that the maximum number of levels is $log_{1/p}n$, so the first term dominates the second term.

(d) **[3 points]** Asymptotically, in terms of $n$ and $p$, what is the expected amount of memory used by the skip list data structure?

**Answer:** Each cell in the skip list takes a constant amount of memory since we need to store the four fields *next*, *above*, *below*, and *key*. The expected number of cells in the skip list is the sum over all elements $A[j]$ of the number of levels that $A[j]$ appears in.

$$E[\text{num cells}] = E[\sum_{j=1}^{n}(\text{num levels that } A[j] \text{ appears in})]$$

By linearity of expectation, we can pull the summation outside of the expectation.

$$E[\text{num cells}] = \sum_{j=1}^{n} E[\text{num levels that } A[j] \text{ appears in}]$$

We found the expected number of levels in part (c), names $E[levels] = 1/(1-p)$. So,

$$E[\text{num cells}] = \sum_{j=1}^{n} 1/(1-p) = n/(1-p).$$

So, the expected amount of memory is $\Theta(n/(1-p))$.

5. [**18 points**] Weighted Selection

Consider some ways to generalize the notion of the median and the $i$th element.

(a) [**1 point**] Do problem 9-2(a) on page 194 of CLRS.

**Answer:** We have that the weighted median $x_k$ satisfies the formulas $\sum_{x_i<x_k} w_i < \frac{1}{2}$ and $\sum_{x_i>x_k} w_i \le \frac{1}{2}$. For $w_i = 1/n$ for $i = 1, 2, \ldots, n$, these formulas reduce to $\sum_{x_i<x_k} w_i = \sum_{x_i<x_k} 1/n = y/n < 1/2$ where $y$ is the number of elements smaller than $x_k$ and $\sum_{x_i>x_k} 1/n = (n-y-1)/n \le 1/2$ where $n-y-1$ is the number of elements greater than to $x_k$. This shows that $y < n/2$ and $y \ge n/2 - 1$. This states that exactly $n/2 - 1$ elements are smaller than $x_k$ for even $n$ and exactly $\lfloor n/2 \rfloor$ elements are smaller than $x_k$ for odd $n$. These is precisely the definition for the *lower* median.

(b) [**2 points**] Do problem 9-2(b) on page 194 of CLRS.

**Answer:** One can calculate the weighted median in $\Theta(n \lg n)$ worst case time using sorting. First, one sorts the array using merge sort which has worst-case running time $\Theta(n \lg n)$. Then we iterate through the array, computing $\sum_{i=1}^{k-1} w_i$, the total weight of the first $k-1$ elements in the array. We find the last element $k-1$ for which $\sum_{i=1}^{k-1} w_i < 1/2$. Then element $k$ satisfies the first equation. It also satisfies the second equation, since $\sum_{i=k+1}^{n} w_i = 1 - \sum_{i=1}^{k} w_i \le 1/2$ since we know that $k-1$ was the *last* element for which the sum was less than $1/2$, so the sum up to $k$ must be $\ge 1/2$. Therefore, $k$ is the weighted median, and we have found it in $\Theta(n)$ time after sorting, so the total amount of time is $\Theta(n \lg n)$.

(c) [**10 points**] Do problem 9-2(c) on page 194 of CLRS.

**Answer:** The main idea is to use deterministic SELECT to compute the regular median in $\Theta(n)$ worst-case time. We can then use this median to partition our array into two (almost) equal halves, and look for the weighted median in either one or the other half. The recurrence will be $T(n) = T(n/2) + \Theta(n)$, which yields $T(n) = \Theta(n)$. In giving the pseudocode we will ignore floors and ceilings for clarity, thus, for instance, the median has index $n/2$.

W-SELECT takes 4 parameters - the array $A$, *left* and *right* boundaries, and weight $w$. We look for an element $x_m$ that will satisfy $\sum_{x_i<x_m} w_i < w$ and $\sum_{x_i>x_m} w_i \le 1 - w$.

```
W-SELECT($A$, $left$, $right$, $w$)
  i. Call SELECT (the deterministic version) to find the median $m$ of $A$.
 ii. Partition $A$ around $m$.
iii. Compute the total weight $W$ of all the elements in the lower partition.
 iv. If $W < w$ but $W + w_{n/2} \ge w$, return $m$.
  v. Else if $W < w$, recursively call W-SELECT($A$, $n/2$, $n$, $w - W - w_{n/2}$).
 vi. Else (if $W > w$), recursively call W-SELECT($A$, 1, $n/2$, $w$).
```

This is a little more general than necessary for weighted median, but it helps in the next part of the problem. To compute the weighted median of an array $A$, we call W-SELECT($A$, 1, $n$, 1/2).

(d) **[5 points]** Consider the following generalization of the $i$th element: the **weighted $i$th element** is the element $x_k$ satisfying

$$\sum_{x_j < x_k} w_j < \frac{i}{n} \text{ and } \sum_{x_j > x_k} w_j \leq \frac{n-i}{n}$$

Modify your algorithms from parts (b) and (c) to compute the weighted $i$th element in, respectively, $\Theta(n \lg n)$ and $\Theta(n)$ worst-cased running time.

**Answer:** We can modify the algorithm in part (b) by simply comparing our sum $\sum_{i=1}^{k-1} w_i$ to $i/n$ instead of $1/2$. The rest of the analysis holds as before. The running time is still dominated by the sort which takes $\Theta(n \lg n)$. The actual selection of the weighted $i^{th}$ element from the sorted array takes time $\Theta(n)$ since it consists of one pass through the sorted array, summing the weights.
In part (c), we can simply call W-SELECT($A$, 1, $n$, $i/n$).

6. **[28 points]** Prisoner's Other Dilemma

An evil computer science professor has kidnapped $n$ of his students and locked them up in a dungeon. Each is kept in a separate cell in complete isolation from others, except that each evening one student is chosen randomly to spend the night in a conference room. The conference room is empty except it contains a switch, which may be toggled ON or OFF. The only way to observe or change the state of the switch is to spend a night in the conference room. Initially, the switch is known to be in the OFF position.

The students are told that once they have all been in the room at least once, one of them needs to tell this to the professor (this can be done when he is transferring the student to the conference room in the evening or back to the cell in the morning). If the claim is true, all students are set free and receive their degrees. If the claim is false, the students will be imprisoned forever and forced to proofread problem sets and papers for food.

The students are given one hour to come up with a joint strategy, after which point they will be taken to their cells and will only be able to communicate by observing and toggling the switch. Luckily, the students have studied algorithms and are able to come up with a solution plan that guarantees that when they do make the claim, it will be correct. Your job will be to analyze the expected time of their solution.

(a) **[10 points]** Consider the following idea: one of the students is designated as the **counter**. His job will be to toggle the switch OFF anytime he enters the room and sees that the switch is set to ON. Everyone else will toggle the switch ON if it's set to OFF, but only once.

Who can make the claim and when? Formally analyze the expected running time of this algorithm. Your answer should be asymptotic in terms of $n$.

**Answer:** $E[time] = \Theta(n^2)$. The **counter** can make the claim that everyone has been in the room once he has toggled the switch to OFF $n-1$ times. Since each student besides the counter sets the switch to ON exactly once, we know that $n-1$ distinct students besides the counter have been in the room. So, the counter

knows that all $n$ students have been in the room exactly once.
We can analyze the expected running time of this algorithm by breaking the sequence of students in the room into periods. We will define two types of periods
$X_i$ - the number of students who enter the room until (and including) the one who sets the switch to ON for the $i^{th}$ time.
$Y_i$ - the number of students who enter the room after the $i^{th}$ student has turned the switch ON until (and including) the counter who turns the switch to OFF.
For example, if student 1 is the counter and there are 4 students, then the following sequence would be broken up into the given periods:
4123431224131
$X_1 = \{4\}$, $Y_1 = \{1\}$, $X_2 = \{2\}$, $Y_2 = \{3, 4, 3, 1\}$, $X_3 = \{2, 2, 4, 1, 3\}$, $Y_3 = \{1\}$
Notice that the periods run $X_1, Y_1, X_2, Y_2, \ldots, X_{n-1}, Y_{n-1}$. After period $Y_{n-1}$ we are finished because the counter has turned off the switch $n-1$ times.

The expected length of time for the algorithm is $E[time] = E[\sum_{i=1}^{n-1}(X_i + Y_i)]$. By linearity of expectation, we can pull the sum outside of the expectation. $E[time] = \sum_{i=1}^{n-1}(E[X_i] + E[Y_i])$.

$X_i$ is the number of students who enter the room until one can set the switch to ON. After $i-1$ students have set the switch to ON, there are $(n-1)-(i-1)$ students who can set it to ON. Since we choose students uniformly at random, the probability at any point that one of these is selected is $(n-i)/n$. Thus, $Pr\{X_i = k\} = (\frac{n-i}{n})^{j-1}(\frac{i}{n})$. This is the geomtric series, and it's expectation is given by $E[X_i] = \frac{n}{n-i}$.

Similarily, we can find the expected value of $Y_i$, the number of students who enter the room after the $i^{th}$ student turns on the switch until the counter enters. The probability that the counter enters is $1/n$. So, $Pr\{Y_i = k\} = (\frac{n-1}{n})^{j-1}(\frac{1}{n})$. Again, this is a geometric distribution with expected value $E[Y_i] = n$. Using these formulas, we can find the expected length of time.

$$\begin{aligned}
E[time] &= \sum_{i=1}^{n-1}\left(\frac{n}{n-i} + n\right) \\
&= n(n-1) + n\sum_{i=1}^{n-1}\frac{1}{n-i} \\
&= n(n-1) + n\sum_{j=1}^{n-1}\frac{1}{j} \\
&= n(n-1) + n\Theta(\ln(n-1)) \\
&= \Theta(n^2)
\end{aligned}$$

(b) **[10 points]** Now consider the following modification to this idea: instead of deciding on the counter ahead of time, the counter will be the first student to enter the room twice. Now depending on what day that occurs, the counter will know how many distinct people have been in the room already. Notice that this means the students need different rules for the first $n$ days of the imprisonment - after that, they can continue using the algorithm from part (a).

How must the students behave during the first $n$ days to make this work? Pay particular attention to what happens on day $n$. Analyze the running time of the modified algorithm - we do not expect a formal analysis this time, but your explanation should be complete and mathematical.

**Answer:** For the first $n$ days everyone should act as follows:

i. The switch remains OFF until someone has been in the room twice. That person turns the switch ON and becomes the counter, setting the count to $j-2$, where $j$ is the day when he enters the room the second time.

ii. If someone other than the counter enters the room and finds the switch OFF, the counter has not been determined yet. Thus, that person will be counted in the $j-2$ and does not touch the switch after day $n$.

iii. If someone other than the counter enters the room and finds the switch ON, the counter has been determined. Therefore, that person has not been counted in $j-2$ and must participate in the algorithm after day $n$. Same goes for anyone who has not entered the room at all during days 1 through $n$.

iv. Finally, the person who is in the room on day $n$ turns the switch OFF prior to beginning the second phase. If the person entering the room on day $n$ finds the switch OFF, then nobody has been in the room twice and that person can make the claim.

After the first $n$ days, everyone should act as in part (a) with the counter determined, except that the people who entered the room when the switch was OFF (befor the counter was determined) no longer participate.

The expected running time of this algorithm depends on how many people we counted during the first $n$ days, call that number $j$. We will first analyze the expected time after the first $n$ days based on $j$. As in part (a), we alternate time periods $X_i, Y_i$, but now since $j$ people have already been counted, we start with period $X_{j+1}$. Therefore, the expected time is given below with the sum starting from $j+1$

$$\begin{aligned} E[time|j] &= \sum_{i=j+1}^{n-1}\left(\frac{n}{n-i}+n\right) \\ &= n(n-j-1)+n\sum_{i=j+1}^{n-1}\frac{1}{n-i} \\ &= n(n-j-1)+n\Theta(\ln(n-j-1)) \end{aligned}$$

Now we will bound the probability of $j$, the number of people counted during the first $n$ days. This is the birthday paradox problem, where the days are the people in the birthday paradox, and the people are the birthdays. We know that when $j = \Theta(\sqrt{n})$, we have $Pr\{\text{someone enters twice in} \le j \text{ days}\} \ge 1/2$. So, we can now bound our expected time.

$$
\begin{aligned}
E[time] &= n + \sum_{j=0}^{n-1} Pr\{j\}E[time|j] \\
&= n + \sum_{j=0}^{c\sqrt{n}} Pr\{j\}E[time|j] + \sum_{j=c\sqrt{n}+1}^{n-1} Pr\{j\}E[time|j] \\
&\ge n + \sum_{j=0}^{c\sqrt{n}} Pr\{j\}E[time|c\sqrt{n}] + 0 \\
&= n + \left(n(n - c\sqrt{n} - 1) + n\Omega(\ln(n - c\sqrt{n} - 1))\right) \sum_{j=0}^{c\sqrt{n}} Pr\{j\} \\
&\ge n + \left(n(n - c\sqrt{n} - 1) + n\Omega(\ln n)\right)(1/2) \\
&\ge n + (1/2)n^2 - (c/2)n^{3/2} - n + \Omega(n \ln n) \\
&= \Omega(n^2)
\end{aligned}
$$

This expected time is also $O(n^2)$, as it's bounded above by $n + E[time|0] = \Theta(n^2)$. Thus, our asymptotic expected time remains $\Theta(n^2)$.

(c) [**8 points**] It is possible to do asymptotically better than your answers in parts (a) and (b). However, there are well-defined lower bounds on the performance of **any** correct algorithm for this problem. For instance, since all students have to have been in the room at the time the claim is made, clearly no correct algorithm can have an expected running time asymptotically less than $\Theta(n)$. Can you come up with a tighter lower bound? Argue that your answer is correct; you may use results proven in class or in the textbook.

**Answer:** No correct algorithm can have a better expected running time than $\Omega(n \lg n)$. As we proved in class with the balls and bins question, the expected amount of time until each bin has at least one ball is $\Theta(n \lg n)$. In this question, the days are balls and the students are bins. We want the expected number time until each students has been in the room for at least one day. So, the expected number of days is $\Theta(n \lg n)$. Any algorithm the students come up with can't have an expected running time faster than the expected time for the condition to actually hold, otherwise it would not be correct. Therefore, the expected running time of any algorithm which will only tell the professor they have all been in the room once they have actually all been there must take at least $\Theta(n \lg n)$ days. Hence, $E[time] = \Omega(n \lg n)$.