CSC263 Fall 2017 Assignment 1

Name: Yuchen Wu, your teammate name
SN: 1002060244, Your teammate students number

Q1. SOLUTION

(a) $T(n)$ is $O(n^2)$.
Since $T(n)$ is the worst case time complexity of executing procedure, the first loop (i.e. for $i = 2$ to n do) runs in $n - 1$ times and each time, the inner loop (i.e. for $j = i$ to n do) has size $n - i$.
The worst case complexity $T(n)$ can be expressed as:

$$T(n) = (n-1) + (n-2) + ... + 1 = \frac{n^2}{2} \in O(n^2)$$

(b) $T(n)$ is $O(n^2)$.
As discussed in part (a), $T(n)$ can be express as:

$$T(n) = (n-1) + (n-2) + ... + 1 = \frac{n^2}{2} \in \Omega(n^2)$$

Q2. SOLUTION

(a) In the best case, Line #2 ("if $A[i] = k$") will be executedd only once.
the first value assigned to i is $n-1$ so that in the first time that Line #2 is being executed, $A[i] = A[n-1]$ and the algorithm is checking whether the last integer in array A is equal to k. Since the integer stored in $A[n-1]$ can be any inumber between 0 and n. Also, we assume that k is an integer whose value satisfies $1 \leq k \leq n$. Therefore, the best case is that $A[n-1] == k$ and Line #2 is only excuted once.

(b) The probability that the best case occurs is $\frac{1}{n+1}$.
Justify my work?

(c) In the worst case, Line #2 ("if $A[i] = k$") will be executedd n times.
the worst case happens when k does not appear in A. In this case, Line #2 is excuted n times and the function returns -1.

(d) since for $A[i]$ we pick an integer from 0 to $i+1$ uniformly, the probability of choosing each integer is equal. Assume a given $1 \leq k \leq n$, then it is impossible for k to appear in $A[i]$ where $i < k-1$.

$$P(A[i] \neq k, i < k-1) = 1$$

The probability of choosing k in $A[i]$ where $k-1 \leq i \leq n-1$ is $\frac{i+1}{i+2}$.

$$P(A[i] \neq k, k-1 \leq i \leq n-1) = \frac{i+1}{i+2}$$

The probability for choosing each number in array$A$ is independent. Therefore, for a given k, the probability that worst case occurs is:

$$\begin{aligned} P(worstcase) &= \Pi_{i=k-1}^{n-1} P(A[i] \neq k) \\ &= \Pi_{i=k-1}^{n-1} \frac{i+1}{i+2} \\ &= \frac{k}{n+1} \end{aligned}$$

(e) In the average case, the last appearance of value k is possible to be found at any $A[i]$ with $k-1 \geq i \geq n-1$. In order to find the average case, we calculate the expected last apperarace of k when the value of k is given.
For a given $k(1 \leq k \leq n-1)$ the probability of $A[i](k-1 \neq i \neq n-1)$ being the last appearance of value$k$ is

$$\begin{aligned} P(A[i]) &= \frac{1}{i+1} \times \Pi_{j=n-1}^{i+1} \frac{j+1}{j+2} \\ &= \frac{1}{n+1} \end{aligned}$$

Also if $A[i](k-1 \geq i \geq n-1)$ is the last appearance of value k, we know that the number of times that Line #2 is excuted (assume m) will be

$$m = n - i$$

Additionally, if value$k$ does not appear in any $A[i](k-1 \geq i \geq n-1)$, then there wont't be any integer in array A with value k. In this case, line #2 will be executed n times. Therefore, the expected position of the last appearance of a given k is

$$\begin{aligned} E(Line\ \#2\ execucuted\times) &= \sum_{m=1}^{n-k+1} \frac{1}{n+1} + n \times (1 - \frac{n-k+1}{n+1}) \\ &= \frac{(n-k+1) \times (n-k+2)}{2 \times (n+1)} + \frac{k}{n+1} \\ &= \frac{(n-k)^2 + 3n - k + 2}{2 \times (n+1)} \end{aligned}$$

Q3. SOLUTION

(a)

Q4. SOLUTION

(a) Generally, I choose to use 2 heaps (1 max-heap and 1 min-heap) to find the median of input numbers. Functional algorithm written C++ is provided below. As seen from in code, when getting the first and second input integers, the function simply output the first number and average of the two imput numbers respectively, which are just the first two required output.This function then stores the greater one between the two into a min-heap and the smaller intege into a max-heap. After that,the functions will check the value of input integer and how many numbers are there in both heaps whenever it gets a new input number. There are six cases (can be five, just to be clearer), as described below.

Use a and b ($a < b$) as the top of max-heap and min-heap respectively, i as the new input integer and number of integers in both heaps are k in total.

(1) k is odd and $a < i < b$

Put i into the heap with less numbers. Output average of the top integer in both heaps.

(2) k is odd and $i \leq a$

Put i into the max-heap. After that, check if the max-heap has more elements than min-heap and if so, extract a from max-heap and insert it into min-heap. Output average of the top integer in both heaps.

(3) k is odd and $i \geq b$

Put i into the min-heap. After that, check if the min-heap has more elements than max-heap and if so, extract b from min-heap and insert it into max-heap. Output average of the top integer in both heaps.

(4) k is even and $a < i < b$

Put i into the min-heap and output b.

(5) k is even and $i \leq a$

Put i into the max-heap and output a.

(6) k is even and $i \geq b$

Put i into the min-heap and output b.

(b) In the worst case, for each new input to this algorithm except the first two numbers, if number of integers in both heaps (k) is odd, then the algorithm performs insert operation twice and extractmax operation once. Since each heap at this time has roughly $k/2$ elements, the time complexity for the entire operation is

$$T(k, kisanoddnumber) = 3log(\frac{k}{2})$$

If number of integers in both heaps (k) is even. then the algorithem only performs insert operation once, so in this case

$$T(k, kisanevennumber) = log(\frac{k}{2})$$

Therfore, the time complexity of the entire algorithm after giving n inputs on average is approximately

$$\begin{aligned}
T(n) &= \sum_{i=3}^{n} 2log(\frac{n}{2}) \\
&\approx \sum_{i=1}^{n} 2log(\frac{n}{2}) \\
&= \sum_{i=1}^{n} 2log(n) - 2nlog(2) \\
&= 2log(n!) - 2nlog(2) \\
&\sim nlog(n) - n + \frac{log(2\pi n)}{2} - 2nlog(2) \in O(nlogn)
\end{aligned}$$