Kunal Patel

CSE 310 Assignment #3
(Max. Points: 30)

Due on: Friday, Oct. 11, 2019, 11:59pm Arizona time

General Instructions:

- This is an individual assignment, please do not collaborate. If there's programming part, make sure that you write every line of your own code. Using code written by someone else will be considered a violation of the academic integrity and will result in a report sent to the Dean's office.
- For all written exercises: **your answer should be clearly typed or written and must be saved in .pdf or .jpg format**. Note: **unreadable answer receives no credits!**
- All assignments must be submitted through the link posted on Blackboard, we do NOT accept any hand-in submissions or submissions sent through emails!
- Submission link will be closed automatically once the due date/time is past and **no late assignment will be accepted**. You will be allowed 3 times to submit the assignment before the due date/time, but we will only grade your last submission.

Objectives

- Heap & operations on a heap.
- Selection algorithms
- Algorithm analysis
- Continue exercises on OpenMP, write simple parallel program.

Questions

1. [3 pts] Write pseudo codes for the function ***Change(A, heapsize, i, newValue)*** that accesses an element at the index *i* in the max-heap *A* (Note that the index of this heap starts from 1), and changes its value to the value of the parameter *newValue*, and also maintains its max-heap property. Note that *newValue* can be larger or smaller than the value of *A*[*i*]. You can use the heap functions/algorithms we discussed in class to implement this function. Also if the value of *i* is out of the range, then it should not change anything in the heap.

```
void Change(A, heapsize, i, newValue)
if (A[i] < newValue)
    HEAP-INCREASE-KEY(A, i, newValue)
else
    BUILD-MAX-HEAP(A)
```

2. [4 pts] Suppose we use **RANDOMZIED-SELECT(*A, p, r, i*)** algorithm (pp. 215 Section 9.2) to select the minimum element of the array $A = \{3, 2, 9, 0, 7, 5, 4, 8, 6, 1\}$. Describe a sequence of partitions that results in a worst-case performance of RANDOMIZED-SELECT.

when doing RANDOMIZED-PARTITION and the random pivot value you get happens to be the largest value, which means you will need to swap with every element in the partition (n-times).

3 2 9 0 7 5 4 8 6 1

// randomly choose 9 to be pivot

3 2 1 0 7 5 4 8 6 9 // swaps each element w/ eachother, runs $O(n)$ times.

// recursive RANDOMIZED-SELECT(A, p, q-1, i)

3 2 1 0 7 5 4 8 6 // choose largest in worst case

3 2 1 0 7 5 4 6 8

// keep going until you have one element which is the smallest since each recursive call the largest is not being included

0 ← runs $O(n)$.

$\Rightarrow$ thus $O(n^2)$ - worst case.

3. [4 pts] In the algorithm we learned in class **SELECT(*A, i*)**, the input elements are divided into groups of 5. Will the algorithm work in linear time if they are divided into groups of 7? Argue that SELECT does not run in linear time if groups of 3 are used.

groups of 7:

$$4\left(\left\lceil \tfrac{1}{2}\left\lceil \tfrac{n}{7}\right\rceil\right\rceil - 2\right) \geq \frac{2n}{7} - 8$$

$$T(n) = \begin{cases} O(1) & \text{if } n < n_0 \\ T(\lceil n/7 \rceil) + T\left(\frac{5n}{7} + 8\right) + O(n) & \text{if } n \geq n_0 \end{cases}$$

$$T(n) \leq c\lceil n/7 \rceil + c\left(\frac{5n}{7} + 8\right) + a_n$$

$$\leq \frac{cn}{7} + c + \frac{5cn}{7} + 8c + a_n$$

$$= \frac{6cn}{7} + 9c + a_n$$

$$= c_n + \left(\frac{-cn}{7} + 9c + a_n\right)$$

$$\leq cn$$

$$= O(n)$$

$$\Rightarrow -\frac{cn}{7} + 9c + a_n \leq 0$$

$$c(n/7 - 9) \geq a_n$$

$$c\left(\frac{n-63}{7}\right) \geq a_n$$

$$c \geq \frac{7a_n}{n-63} \quad \text{when } n > 63.$$

assume $n_0 \geq 126$ $\quad \frac{n}{(n-63)} \leq 2$

then $c \geq 14a$

choosing any integer greater than 63 and c accordingly, therefore the worst-case running time is linear.

groups of 3:

$$2\left(\left\lceil \tfrac{1}{2}\left\lceil \tfrac{n}{3}\right\rceil\right\rceil - 2\right) \geq \frac{n}{3} - 4$$

$$T(n) = T(\lceil n/3 \rceil) + T(2n/3 + 4) + O(n)$$

$$T(n) \geq c\lceil n/3 \rceil + c(2n/3 + 2) + a_n$$

$$\geq cn/3 + c + 2cn/3 + 2c + a_n$$

$$= cn + 3c + a_n$$

$$\geq cn$$

holds for any $c > 0$

4. [12 pts] Given an unsorted array of n unique integers, you need to return the k smallest of them in sorted order. You come up with three algorithms to solve this problem: They are:

- Algorithm **A**: Sort the array in increasing order, and list the first k integers.
- Algorithm **B**: Build a min-heap from these n integers, and then call Extract-Min k times.
- Algorithm **C**: Use the linear time selection algorithm to find the kth smallest integer, then partition the array about that number, and finally sort these k smallest numbers.

1) [3 pts] Let $T_A(n, k)$ denote the worst-case running time of Algorithm A. Analyze $T_A(n, k)$ using the big-O notation, in terms of n and k. Justify your answer.

Using insertion sort as Algorithm A, you get worst-case running time as $O(n^2)$, and returns first K integers returning the first smallest would be $O(1)$, and first K would be $O(K)$ and so for algorithm A you would get the worst-case running time as $O(n^2) + O(K)$

2) [3 pts] Let $T_B(n, k)$ denote the worst-case running time of Algorithm B. Analyze $T_B(n, k)$ using the big-O notation, in terms of n and k. Justify your answer.

Sorting using min-heap takes $O(n \lg n)$ as the worst case, and returns the k^{th} smallest element which calls Extract-Min K times which has a running time of $O(K)$. So algorithm B has a worst case running time of $O(n \lg n) + O(K)$

3) [3 pts] Let $T_C(n, k)$ denote the worst-case running time of Algorithm C. Analyze $T_C(n, k)$ using the big-O notation, in terms of n and k. Justify your answer.

Sorting using the Select algorithm takes $O(n)$ in the worst case, as it runs in linear time. Then, partitioning the array about the k^{th} smallest integer will run in $O(n)$ time. And sorting the k smallest numbers will be $O(K)$, thus algorithm C worst-case running time is $O(n) + O(K)$

4) [3 pts] Based on your analysis in the above, which algorithm would you choose to find the k smallest integers in sorted order, and why?

I would choose the algorithm C because it has fastest running time for the worst-case. Only if you have a large array, if you have a small array algorithm would be more efficient because of the growth rate of lg:

5. [7 pts] Continue reading the notes on OpenMP (file *OpenMP.pdf*), see the attached prob5.cpp file, and complete the program by writing code to compute a sum of randomly generated numbers by:
a) [1 pt] Complete ***sumWithLoop*** function to compute the sum of elements in the array using a *for* loop sequentially. Include your code for this function in the file that you will be submitting.

b) [1 pt] Complete ***sumWithLoop_OMP*** function to compute the sum of elements in the array using a *for* loop *in parallel* (using OMP). Include your code for this function in the file that you will be submitting.

c) [1 pt] Complete ***sumRec*** function to compute the sum of elements in the array recursively by splitting the parameter array into halves every time. Include your code for this function in the file that you will be submitting.

d) [1 pt] Compete ***SumRec_OMP*** function to compute the sum of elements in the array recursively by splitting the parameter array into halves every time and executing two recursive calls in parallel (using OMP). Include your code for this function in the file that you will be submitting.

e) [3 pts] Using 4 threads, execute your program using 100, 10000, 1000000, and 10^8 as the values of *n* (the number of randomly generated numbers and also the array size). Then record the execution time for each of the four functions above in a table format:

Execution time	***n* =100**	***n* =10000**	***n* =1000000**	***n* =100000000**
sumWithLoop	0.000014	0.000034	0.003313	0.315241
sumWithLoop_OMP	0.000239	0.000230	0.001064	0.075738
sumRec	0.000002	0.000107	0.010765	1.130387
SumRec_OMP	0.000679	0.013482	1.120670	116.897129

Note: after completing your functions, you can compile & run it in *general.asu.edu* as:

```
$ g++  -fopenmp -o prob5 prob5.cpp
```

To set the number of threads to 4, type the following before you execute the prob5

```
$ export OMP_NUM_THREADS=4
```

To execute, type:

```
$./prob5
```

Note for Question #5: For this question (#5), you need to submit your finished prob5.cpp on Canvas!