King Fahd University of Petroleum & Minerals

College of Computer Science and Engineering

Information and Computer Science Department

Second Semester 152 (2015/2016)

ICS 202 – Data Structures

Major Exam 1

Sunday, Feb 28th, 2016

Time: 90 minutes

Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID# |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Section 01Dr. Emad |  | Question # | CLO | Max Marks | Marks Obtained |
|  |  | 1 | 3 | 15 |  |
| 09-09:50am |  | 2 | 3 | 15 |  |
|  |  | 3 | 3 | 20 |  |
|  |  | 4 | 2 | 30 |  |
| Section 02Dr. Emad |  | 5 | 2 | 20 |  |
| 10-10:50am |  | Total |  | 100 |  |

**Instructions**

1. **Write your name and ID in the respective boxes above and circle your section.**
2. **This exam consists of 6 pages, including this page, plus one reference sheet, containing 5 questions.**
3. **You have to answer all 5 questions.**
4. **The exam is closed book and closed notes. No calculators or any helping aids are allowed.**
5. **Make sure you turn off your mobile phone and keep it in your pocket if you have one.**
6. **The questions are not equally weighed.**
7. **The maximum number of points for this exam is 100.**
8. **You have exactly 90 minutes to finish the exam.**
9. **Make sure your answers are readable.**
10. **If there is no space on the front of the page, feel free to use the back of the page. Make sure you indicate this in order not to miss grading it.**

**Q.1 (15 points) Complexity Analysis and Recursion**

* 1. **(5 points) Prove that n2/2 - 2n is Θ (n2)**

**0 < c1 < ¼ < c2 at N=8**

* 1. **(10 points) Write a recursive function that finds the sum value of an array of integers.**

**int Sum(int[] x, int index) {**

 **if (index == 0)**

 **return x[0];**

 **else**

 **return x[index]+ Sum(x, index – 1);**

**}**

**Q2. (15 points) Consider the following summarized version of Algorithm mergeSort:**

 static void mergeSort(int[] A) {

 if (A.length > 1) {

 int q = A.length/2;

 int[] leftArray = Arrays.copyOfRange(A, 0, q);

 int[] rightArray = Arrays.copyOfRange(A,q+1,A.length);

 mergeSort(leftArray);

 mergeSort(rightArray);

 A = merge(leftArray,rightArray);

 }

 }

**Assuming that the merge function in the last line of the algorithm will carry out n – 1 element comparisons when Algorithm mergeSort is called to sort n elements**

1. **(5 points) Derive the recurrence equation describing the worst-case time** **complexity of the algorithm in terms of the number of element comparisons.**

**T(n) = 2T(n/2) + (n-1) , T(1) = 0**

1. **(10 points) Solve the recurrence equation in part a, and express it in terms of Big O() notation**

**T(n) = 2T(n/2) + (n-1)**

**= 2[ 2T(n/4) + (n/2 -1)] + (n-1)**

**= 4T(n/4) + 2n – 3**

**= 8T(n/8) + 3n – 7**

**= 2k T(n/2k) +kn – (2k - 1)**

**Stop at step 1, when n/2k = 1, when k = log2 n**

**Then T(n) = n T(1) + n log2 n – (n -1) = n log2 n – (n -1) which is O(n log2 n )**

**Q.3 (20 points) For the following piece of code:**

1. public static int complex(float [] array, int n , int m) {
2. for (int i=1; i<=n; i\*=2) {
3. for (int j=1; j<i; j++) {
4. if (array[i] > array[j])
5. array[j] = array[j]+1;
6. }
7. }
8. }
9. **(2 points) Which line of code do you think will be a good representative of the total cost of the algorithm? (i.e., which line of code do you think we should count the number of times it is executed?)**

line 4

1. **(10 points) Represent the cost of running this code in terms of [a] summation[s]. Then evaluate the summation[s] in the worst case.**

 (Outer loop) *i* : 1 , 2 , 22 , 23 ,…, 2*k* = *n*  (2k=n -> *k* = log2 *n*)

 Hence, we can use *k* where *i* = 2*k* *k* : 0 , 1 , 2 , 3, …, *k* = log2 *n*

 (Inner loop) *j* : 1 , 2, ,…, i-1 and i=2*k*

$$\sum\_{k=0}^{log n}\sum\_{j=1}^{i-1}1= \sum\_{k=0}^{log n}i-1$$

$$\sum\_{k=0}^{log n}2^{k}-1= \frac{2^{log n +1}-1}{2-1}-\left(log n +1 \right)$$

**=**

**= (2n -1) – log n -1**

**=O(n)**

1. **(8 points) In the best-case analysis, we calculate lower bound on running time of an algorithm. We must know the case that causes minimum number of operations to be executed. Which case of the array do you think will give us the best case?**

**When if-statement value is always false, i.e. if the array is sorted in decreasing order.**

**Q.4 (30 points)** Consider the SLL class definition as seen in the lectures:

**public class SLL<T> {**

 **protected SLLNode<T> head, tail;**

 **public SLL() {**

 **head = tail = null;**

 **}**

 **public boolean isEmpty() {**

 **return head == null;**

 **}**

 **...**

**}**

Implement a method called **kSmallest** that places the first k smallest values, sorted in ascending order, in the beginning of the linked list.

 For example, if a linked list of Integer elements is containing the following values,

25

7

14

-1

6

34

2

**null**

**head**

**tail**

After a call to:

**KSmallest(3)**, the single list becomes:

-1

2

6

25

7

14

34

**null**

**head**

**tail**

**public void kSmallest(int k)**

**{**

 **if(isEmpty())**

 **return**

 **if (k > length())**

 **{ System.out.println("Invalid k value");**

 **return;**

 **}**

 **Stack<T> tempStack = new Stack<T>();**

 **for(int j=1; j<= k; j++)**

 **{**

 **T min = head.info;**

 **SLLNode<T> temp = head;**

 **SLLNode<T> prevTemp = head;**

 **while (temp != null)**

 **{**

 **if(temp.info < min)**

 **{ min = temp.info;**

 **prevMin = prevTemp;**

 **}**

 **prevTemp = temp;**

 **temp = temp.next;**

 **}**

 **tempStack.push(min);**

 **delete(min);**

 **}**

 **for(int j=1; j<=k; j++)**

 **addToHead(tempStack.pop());**

 **}**

**Q.5 (20 points)** Consider the DLL and DLLNode classes:

**public class DLLNode<T> {**

 **public T info;**

 **public DLLNode<T> next, prev;**

 …..

}

**public class DLL<T> {**

 **private DLLNode<T> head, tail;**

 **…….**

**}**

Provide a method **selectionSort** in the class DLL<T> which sorts the nodes in ascending order.

**public void selectionSort()**

**{**

 **if (head==tail)**

 **return;**

 **for(int i=1; i<n; i++)**

 **{**

 **DLLNode<T> temp;**

 **for(int j = 1,temp=head; j<i; j++,temp=temp.next);**

 **DLLNode<T> tempMin = temp, temp0 = temp;**

 **while(temp!=null)**

 **{**

 **if(temp.info < tempMin.info)**

 **tempMin = temp;**

 **temp = temp.next;**

 **}**

 **swap(temp0, temp);}**