**Department of Mathematics**

**Ahmadu Bello University, Zaria**

**First Semester Examination 2010/2011 Session**

**COSC401:Algorithm and Complexity Analysis**

**Instructions: Answer Any Four Questions. Time Allowed: 2 Hours**

1. The time complexity of an algorithm is generally evaluated by estimating the total number of basic operations to be executed rather than performing an empirical analysis which consists of obtaining the exact running time from it corresponding running code, by running the program on a computer and evaluate the time it has taken to complete it execution. According to you why not just perform the empirical analysis of an algorithm to determine its running time?
2. Explain why the statement, "The running time of algorithm *A* is at least *O*(*n*2)," is meaningless.
3. Quicksort algorithm developed by C. A. R. Hoare sorts by employing a divide and conquer strategy to divide a list into two sub-lists. Its steps are:
* Pick an element, called a pivot, from the list.
* Reorder the list so that all elements with values less than the pivot come before the pivot, while all elements with values greater than the pivot come after it (equal values can go either way). After this partitioning, the pivot is in its final position.
* Recursively sort the sub-list of lesser elements and the sub-list of greater elements.
1. State when does the worst case occur while sorting a list using quick sort.
2. Obtain the worst case recurrence equation and hence evaluate it time complexity
3. State when does the best case occur while sorting a list in quick sort .
4. Obtain the best case recurrence equation and hence evaluate it time complexity
5. Insertion sort can be expressed as a recursive procedure as follows. In order to sort *A*[1.. *n*], we recursively sort *A*[1 .. *n* -1] and then insert *A*[*n*] into the sorted array *A*[1.. *n* - 1].
6. Obtain a recurrence equation for the running time of this recursive version of insertion sort.
7. Solve the obtained equation to determine the running time.
8. The Tower of Hanoi or Towers of Hanoi , also called the Tower of Brahma , consists of three rods, and a number of disks of different sizes which can slide onto any rod. The puzzle starts with the disks in a neat stack in ascending order of size on one rod, the smallest at the top, thus making a conical shape. The objective of the puzzle is to move the entire stack to another rod, obeying the following rules:
* Only one disk may be moved at a time.
* Each move consists of taking the upper disk from one of the rods and sliding it onto another rod, on top of the other disks that may already be present on that rod.
* No disk may be placed on top of a smaller disk.
1. Obtain the recurrence equation of the Tower of Hanoi that define the running time of the algorithm
2. Solve the obtained recurrence equation to determine the running time
3. Explain why the complexity of the solution of the tower of Hanoi problem is said not to be of class P
4. Given the following list

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 6 | 2 | 0 | 4 | 3 | 8 | 5 | 9 | 7 |

1. Show how the list will be sorted using bubble sort algorithm
2. Show how the list will be sorted using selection sort
3. Consider sorting *n* numbers stored in array *A* by first finding the largest element of *A* and exchanging it with the element in *A*[1]. Then find the second largest element of *A*, and exchange it with *A*[2]. Continue in this manner for the first *n* - 1 elements of *A*.
4. Write pseudo-code for this algorithm
5. Why does it need to run for only the first *n* - 1 elements, rather than for all *n* elements?
6. When does the best case and the worst case occur in this algorithm?
7. Obtain the best-case and worst-case running times of the algorithm.
8. Solve the following recurrence equation

i. T (n) = 3T (n/2)+ n2 ii. T (n) = 4T (n/2)+ n2

iii. T (n) = T (n/2) + 2n iv. T (n) = 16T (n/4)+ n

1. T (n) = 2T (n/2)+ n log n
2. We can extend our notation of *O* to the case of two parameters *n* and *m* that can go to infinity independently at different rates. For a given function *g*(*n*, *m*), we denote by *O*(*g*(*n*, *m*)) the set of functions *O*(*g*(*n*, *m*)) = {*f*(*n*, *m*): there exist positive constants *c*, *n*0, and *m*0 such that 0 ≤ *f*(*n*, *m*) ≤ *cg*(*n*,*m*) for all *n* ≥ *n*0 and *m* ≥ *m*0}.
3. Give the corresponding definition for Ω(*g*(*n*, *m*))
4. Give the corresponding definition for Θ(*g*(*n*, *m*))
5. Let *f(n,m)=2nm2+3mn+5* and *g(n,m)=n3m3 +m+2*, show that *f(n,m)= Og(n,m)*
6. Let *f*(*n*) and *g*(*n*) be asymptotically positive functions. Prove or disprove each of the following conjectures.
7. *f*(*n*) = *O*(*g*(*n*)) implies *g*(*n*) = *O*(*f*(*n*)).
8. *f*(*n*) + *g*(*n*) = Θ(min(*f*(*n*), *g*(*n*))).
9. *f*(*n*) = *O*(*g*(*n*)) implies *g*(*n*) = Ω(*f*(*n*)).
10. *f*(*n*) = Θ(*f*(*n*/2)).
11. *f*(*n*) + *o*( *f*(*n*)) = Θ(*f*(*n*)).
12. Argue whether the solution to the recurrence *T* (*n*) = *T* (*n*/3) + *T* (2*n*/3) + *cn*, where *c* is a constant, is Ω(*n* lg *n*) by appealing to a recursion tree.
13. Explain the operation of Greedy algorithm and state the properties of problems for which it may work well.
14. Let *f*(*n*) and *g*(*n*) be asymptotically nonnegative functions. Using the basic definition of Θ-notation, prove that max(*f*(*n*), *g*(*n*)) = Θ(*f*(*n*) + *g*(*n*)).
15. The Longest Common Subsequence (LCS) Problem is to find the longest subsequence common to all sequences in a set of sequences (often just two). Note that subsequence is different from a substring. For example, the LCS(KADUNA, KANO)= KAN, and is of length 3.
16. State the recurrence relation that define the optimal substructure of an LCS
17. State an algorithm that compute the length of the LCS based on the recurrence defined in (i).
18. Given two string KADUNA, KAKNO show how the length of their longest common subsequence can be obtained using dynamic programming.