**Department of Mathematics**

**Ahmadu Bello University, Zaria**

**First Semester Examination 2014/2015 Session**

**COSC401: Algorithms and Complexity Analysis**

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

1. What do you understand by brute force analysis of algorithm? State its strengths and weakness
2. Determine the minimum number of character comparisons made by the brute-force algorithm in searching for the pattern GANDHI in the text THERE\_IS\_MORE\_TO\_LIFE\_THAN\_INCREASING\_ITS\_SPEED

Assume that the length of the text which is 47 characters long is known before the search starts.

1. Consider the following algorithm.

**ALGORITHM** *Secret(A*[0*..n* − 1]*)*

//Input: An array *A*[0*..n* − 1] of *n* real numbers

*x*←*A*[0];

*y*←*A*[0]

**for** *i* ←1 **to** *n* − 1 **do**

**if** *A*[*i*]*< x*

*x*←*A*[*i*]

**if** *A*[*i*]*> y*

*y*←*A*[*i*]

**return** *x* − *y*

1. What does this algorithm compute?
2. What is its basic operation?
3. What is the efficiency class of this algorithm?
4. Given the following table representing the frequency of each symbol in a given text.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Symbol | A | B | C | D | - |
| Frequency | 0.35 | 0.1 | 0.2 | 0.2 | 0.15 |

1. Construct its Huffman coding tree.
2. Determine the codeword for each symbol
3. Encode ABAC using the code of question (a).
4. Decode 10001011100100 using the code of question (a).

Consider the following recursive algorithm.

**ALGORITHM** *Riddle(A*[0*..n* − 1]*)*

//Input: An array *A*[0*..n* − 1] of real numbers

**if** *n* = 1 **return** *A*[0]

**else** *temp*←*Riddle(A*[0*..n* − 2]*)*

**if** *temp* ≤ *A*[*n* − 1] **return** *temp*

**else return** *A*[*n* − 1]

1. What does this algorithm compute?
2. Set up a recurrence relation for the algorithm’s basic operation count
3. Solve the recurrence obtained in (ii).
4. What do you understand by the following concepts
5. Class P
6. Class NP
7. Solve the following recurrence equations
8. T (n) = 2T (n/2)+ n log n
9. T(n)=2T(n-1) +1
10. Let *f*(*n*) and *g*(*n*) be asymptotically positive functions. Prove the following assertions by using the definitions of the notations involved, or disprove them by giving a specific counterexample.
11. *f*(*n*) = *O*(*g*(*n*)) implies *g*(*n*) = *O*(*f*(*n*)).
12. *f*(*n*) + *g*(*n*) = Θ(min(*f*(*n*), *g*(*n*)).
13. *f*(*n*) = *O*(*g*(*n*)) implies *g*(*n*) = Ω(*f*(*n*)).
14. Explain what you understand by algorithms and explain why algorithms need to be analyzed
15. 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*.
16. Write pseudo-code for this algorithm
17. When does the best case and the worst case occur in this algorithm?
18. What is the efficiency class of this algorithm?
19. Use the algorithm to sort the following list: 1, 8, 6, 5, 3, 7, 4
20. Given the following list, 1, 8, 6, 5, 3, 7, 4
21. Construct a heap for the list by successive key insertions
22. Sort the list by using Heapsort
23. Consider the algorithm for the sorting problem that sorts an array by counting, for each of its elements, the number of smaller elements and then uses this information to put the element in its appropriate position in the sorted array:

**ALGORITHM** *ComparisonCountingSort(A*[0*..n* − 1]*)*

//Sorts an array by comparison counting

//Input: Array *A*[0*..n* − 1] of orderable values

//Output: Array *S*[0*..n* − 1] of *A*’s elements sorted in nondecreasing order

(1) for *i* ←0 to *n* − 1 do

(2) *Count*[*i*]←0

(3) for *i* ←0 to *n* − 2 do

(4) for *j* ←*i* + 1 to *n* − 1 do

(5) if (*A*[*i*]*<A*[*j* ])

(6) *Count*[*j* ]←*Count*[*j* ]+ 1

(7) else *Count*[*i*]←*Count*[*i*]+ 1

(8) for *i* ←0 to *n* − 1 do

(9) *S*[*Count*[*i*]]←*A*[*i*]

(10) return *S*

1. Apply this algorithm to sorting the list 60, 35, 81, 56, 14, 47.

Hint: specify the content of array count[0..n-1] at the end of the loop of line (1), and after each pass of the loop of line(3) and finally, the content of the array S[0..n-1] after the loop of line (8) is executed.

1. Determine the time efficiency class of this algorithm. Show all details of your answer.
2. Is there any difference on the performance of this algorithm depending on whether the initial list is sorted or not?
3. The knapsack problem is stated as follow: Given n items of known weights w1, w2, . . . , wn and values v1, v2, . . . , vn and a knapsack of capacity W, find the most valuable subset of the items that fit into the knapsack.

The following table shows a small instance of the Knapsack problem having five items with their respective weights and values.

|  |  |  |
| --- | --- | --- |
| **Item No.** | **Item Weight** | **Item Value** |
| 1 | 2 | $15 |
| 2 | 3 | $20 |
| 3 | 1 | $10 |
| 4 | 2 | $12 |

Assuming a knapsack of capacity of 5 units of weight, answer the following questions.

1. find the most valuable subset of the items that fit into the knapsack using exhaustive search.
2. find the most valuable subset of the items that fit into the knapsack using greedy algorithm.
3. find the most valuable subset of the items that fit into the knapsack using Dynamic Programming.