QNo.1  What is heap and what is heap order? (Mark2)
Answer:-
The heap is the section of computer memory where all the variables created or initialized at runtime are stored. The heap order property: in a (min) heap, for every node X, the key in the parent is smaller than or equal to the key in X. 
Ref:  Handouts Page no. 40

QNo.2  Quick sort such that sort the array in to non-increasing order? (Mark2)
Answer:-

Quick sorting, an array A[1..n] of n numbers We are to reorder these elements into increasing (or decreasing) order. More generally, A is an array of objects and we sort them based on one of the attributes - the key value. The key value need not be a number. It can be any object from a totally ordered domain. Totally ordered domain means that for any two elements of the domain, x and y, either x < y, x = y or x > y.
Ref:  Handouts Page no. 40

QNo.3  Draw the cost table for chain multiplication problem with initial states(Mark3)
Answer:-
or (A1A2)(A3A4 . . .An) 
or (A1A2A3)(A4 . . .An) 
. . . . . . 
or (A1A2A3A4 . . .An−1)(An)
Ref:  Handouts Page no. 90

QNo.4  we can avoid unnecessary repetitions for recursive calls? (Mark3)
Answer:-
We can avoid these unnecessary repetitions by writing down the results of recursive calls and looking them up again if we need them later. This process is called memorization
Ref:  Handouts Page no. 49

Worst case for edit distance algorithm? What is the simple change that can change the worst case time ? 5 marks
Answer:-
Analysis of DP edit distance

There are $\Theta(n^2)$ entries in the matrix. Each entry $E(i,j)$ takes $\Theta(1)$ time to compute. The total running is $\Theta(n^2)$. Recursion clearly leads to the same repetitive call pattern that we saw in Fibonacci sequence. To avoid this, we will use the DP approach. We will build the solution bottom-up. We will use the base case $E(0,j)$ to fill first row and the base case $E(I,0)$ to fill first column. We will fill the remaining $E$ matrix row by row.

Ref: Handouts Page no. 14

Describe an efficient algorithm to find the median of a set of 106 integers; it is known that there are fewer than 100 distinct integers in the set

Solution:

Step1: Start
Step2: Find the 100 distinct numbers among $10^6$ integers.
Step3: Sort the 100 distinct numbers
Step4: Count the distinct numbers
Step5: If count is odd, middle number is the median
Step6: If count is even, add the middle two numbers then divide by 2, the result is the median

Ref: Handouts Page no. 34

What is the formula for generating Catalan numbers?

Solution

Equation (22) is a recurrence relation.

$C_{n+1} = C_n \cdot \frac{[2(2n+1)]}{(n+2)}$

we have the values of $n$ in one column and the values of $C_n$ in another, then to put this formula in Excel, on the $(n+1)$-th row just replace $C_n$ and $n$ with the appropriate cells from the previous row.

Ref: Handouts Page no. 85

What are Catalan numbers? Give the formula.

Catalan numbers form a sequence of natural numbers that occur in various counting problems, often involving recursively defined objects

Formula is $C(n) = 2n \cdot C_n / (n+1)$

Ref: Handouts Page no. 85
Q-Write a pseudo code Fibonacci With memorization? -- (3)

Sol
MEMOFIB(n)
1 if (n < 2)
2 then return n
3 if (F[n] is undefined)
4 then F[n] MEMOFIB(n − 1) + MEMOFIB(n − 2)
5 return F[n]

Ref: Handouts Page no. 12, 74

Q – Write Down the steps of Dynamic programming (5)
Dynamic programming is essentially recursion without repetition. Developing a dynamic programming algorithm generally involves two separate steps:

- Formulate problem recursively. Write down a formula for the whole problem as a simple combination of answers to smaller sub problems.
- Build solution to recurrence from bottom up. Write an algorithm that starts with base cases and works its way up to the final solution.

Dynamic programming algorithms need to store the results of intermediate sub problems. This is often but not always done with some kind of table. We will now cover a number of examples of problems in which the solution is based on dynamic programming strategy.

Ref: Handouts Page no. 74 – 77

How we build heap
We build a max heap out of the given array of numbers A[1..n]. We repeatedly extract the maximum item from the heap. Once the max item is removed, we are left with a hole at the root.

Ref: Handouts Page no. 41

Write Pseudo code for KNAPSACK algorithm? 5 marks
Solution:
KNAPSACK (n, W)
1 for w=0,W
2 do V[0,w]←0
3 for i=0,n
4 do V[i,0]←0
5 for w=0,W
do if \((w_i \leq w \& v_i + V[i-1,w] + V[i-1,w]) \geq \text{V}[i-1,w])\)
then \(V[I,w] \leftarrow v_i + V[i-1,w]\)
else \(V[i,w] \leftarrow V[i-1,w]\)
The time complexity is clearly \(O(n,W)\), it must be cautioned that as \(n\) and \(W\) get large, both time and space complexity become significant.

**Spelling correction in edit distance? 3 marks**
A better way to display this editing process is to place the words above the other:

\[\text{S D I M D M} \]
\[\text{M A - T H S} \]
\[\text{A - R T - S} \]

**THE FIRST WORD HAS GAP FOR EVERY INSERTION (1) AND THE SECOND WORD HAS A GAP FOR EVERY DELETION (D). MATHES (M) DO NOT COUNT. THE EDIT TRANSCRIPT IS DEFINED AS A STRING OVER THE ALPHABET \(M,S,I,d\) THAT DESCRIBES A TRANSFORMATION OF ONE STRING INTO OTHER. FOR EXAMPLE**

\[\text{S D I M D M} \]
\[1+1+1+0+1+0+=4 \]

Ref: Handouts Page no. 77

**Differentiate b/w Bubble sort, insertion sort and selection sort? 3 marks**

**SOLUTION:**
Bubble sort: scan the array. Whenever two consecutive items are found that are out of order, swap them. Repeat until all consecutive items are in order.
Insertion sort: assume that \(A[1..i-1]\) have already been sorted. Insert \(A[i]\) into its proper position in this sub array. Create this position by shifting all larger elements to the right.
Selection sort:
Assume that \(A[1..i-1]\) contain the \(i-1\) smallest elements in sorted order. Find the smallest in \(A[i..n]\) swap it with \(A[i]\).

Ref: Handouts Page no. 54

**Write down the steps of dynamic programming strategy. (2 marks)**
Solution:
Developing a dynamic programming algorithm generally involves two separate steps:
1. formulate problem recursively.
Write down a formula for the whole problem as a simple combination of answers to smaller sub problems.
2. Build solution to recurrence from bottom up:

Ref: Handouts Page no. 75
Solve the recursion problem. (5 marks.)
Solution:
Recursion clearly leads to the same repetitive call pattern that we saw in Fibonacci sequence. To avoid this, we will use the DP approach. We will build the solution bottom-up. We will use the base case $E(0,j)$ to fill first row and the base case $E(I,0)$ to fill first column. We will fill the remaining $E$ matrix row by row.
If we trace through the recursive calls to $MemoFib$, we find that array $F[]$ gets filled from bottom up...i.e., first $F[2]$, then $F[3]$, and so on, upto $F[n]$. we can replace recursion with a simple for-loop that just fills up the array $F[]$ in that order.

- We are given an array of n elements of $x_1, x_2, x_3, ..., x_n$, suggest best sorting algorithm of order $O(n)$. (5 marks).
Solution:
The main shortcoming of counting sort is that it is useful for small integers, i.e., $1...k$ where $k$ is small. If this were a million more, the size of the rank array would also be a million. Radix sort provides a nice work around this limitation by sorting numbers one digit at a time.

What are the two steps generally involved while developing dynamic programming algorithm. (2)
Solution:
Developing a dynamic programming algorithm generally involves two separate steps:
- **Formulate problem recursively.**
  Write down a formula for the whole problem as a simple combination of answers to smaller subproblems.
- **Build solution to recurrence from bottom up.**
  Write an algorithm that starts with base cases and works its way up to the final solution.
Ref: Handouts Page no. 75

How we build heap? (2)
Solution:
We build a max heap out of the given array of numbers $A[1..n]$. We repeatedly extract the maximum item from the heap. Once the max item is removed, we are left with a hole at the root.
Ref: Handouts Page no. 41

What are the applications of edit distance technique? Name any three (3)
Solution:
Spelling Correction
Plagiarism Detection
Computational Molecular Biology
Solve: \( T(n) = (T(q - 1) + T(2 - q) + 2) \) \( (3) \)

**What is the worst case running time for the bucket sort? What simple change is required in the algorithm to preserve its linear expected running time and makes it worst case time \( \Theta(n \log n) \)? \( (5) \)**

**Solution:**
The worst case for bucket sort occurs when all inputs falls into single bucket, for example. Since we use insertion sort for sorting buckets and insertion sort has a worst case of \( O(n^2) \), the worst case runtime for bucket sort is \( O(n^2) \).

By using an algorithm with worst case runtime of \( O(n \log n) \) instead of insertion sort for sorting buckets, we can ensure that worst case is \( O(n \log n) \) without affecting the average case behavior.

To see that the worst case is still \( O(n \log n) \), let's consider a case where \( n \) data are distributed among two buckets, \( a \) elements in one bucket and \( n - a \) in the other. Since we use \( O(n \log n) \) sorting algorithm in each bucket, the run time for each sort is, \( k(a) + c_2 \) and \( k(n - a) \log(n - a) + c_2 \), where \( k; c_2 \) are positive constants. The total runtime is \( k(a) + k(n - a) \log(n - a) + 2c_2 \). This quantity attains its maximum value when \( a = 0 \) or \( a = n \) and the maximum value is \( kn \log n + 2c_2 \). Thus total runtime is still \( O(n \log n) \). It is clear from this that maximum running cost occurs when data are in single bucket instead of spread in two buckets. Extending this argument, we can see that worst case for the hash table occurs when all inputs hash into the same bucket. (We also note that the expressions obtained are basically convex combinations of \( n \log n \) which is a convex function and then Jensen's rule can be applied to arrive at the same conclusion).

**Ref:** [http://classes.soe.ucsc.edu/cmps101/Spring11/hw/hw3sol.pdf](http://classes.soe.ucsc.edu/cmps101/Spring11/hw/hw3sol.pdf)

**Given an unordered list of \( n \) \( x_0, x_1, x_2, \ldots, x_n \) and elements is common, if there are at least \( n/5 \) copies of it. We want to identify all the common numbers in our list. Give \( O(n \log n) \) to solve the problem. \( (5) \)**

**Solution:**
We define \( R_n \) to be the set of ordered \( n \)-tuples of real numbers,
\[ R_n := \{ (x_1, x_2, \ldots, x_n) : x_1, \ldots, x_n \in R \}. \]
The elements of \( R_n \) are called vectors. Given a vector \( x = (x_1, \ldots, x_n) \), the numbers \( x_1, \ldots, x_n \) are called the components of \( x \).
You are already quite familiar with \( R_n \) for small values of \( n \).

**Ref:** [http://www.math.rice.edu/~hassett/teaching/221fall05/linalg1.pdf](http://www.math.rice.edu/~hassett/teaching/221fall05/linalg1.pdf)
Find the out cost $A1=5 \times 4 \ A2=4 \times 6 \ A3=6 \times 2$ (2 marks)

Solution:
For Instance

\[
\begin{align*}
A1 & = 5 \times 4 \\
A2 & = 4 \times 6 \\
A3 & = 6 \times 2 \\
A4 & = 2 \times 7
\end{align*}
\]

Hence

\[
A1 \times (A2 \ A3) \times A4 = ((5 \times 4 \times 2) + (4 \times 6 \times 2)) + 2 \times 7 \times 5
\]

\[
= 40 + 48 + 70 = 88 + 70 = 158
\]

Here optimal sequence is $A1 \ (A2 \ A3) \ A4$. For this cost 158 which is optimal the optimal sequence is $a1 \times (a2 \ a3) \ a4$

Ref:: http://books.google.com.pk/books?id=9mBJ0CpKXdC&pg=SA11-PA37&lpg=SA11-PA37&dq=Find%20the%20out%20cost%20A1%3D5%20x%204%20%20A2%3D%204%20x%206%20%20A3%3D%206x2&source=bl&ots=xYZLnvJv6zx&sig=C_u-HLVZ-iQCjvwNpG2hhkosCf4&hl=en&sa=X&ei=i_TEUInCMJQyhoDYaAw&redir_esc=y#v=onepage&q=Find%20the%20out%20cost%20A1%3D5%20x%204%20%20A2%3D%204%20x%206%20%20A3%3D%206x2&f=false

How to construct an optimal solution for 0/1 knapsack problem?

Construction of optimal solution: Construct an optimal solution from computed information. Let us try this: If items are labelled $1, 2, \ldots, n$, then a subproblem would be to find an optimal solution for $S_k = \text{items labelled } 1, 2, \ldots, k$. This is a valid subproblem definition. The question is: can we describe the final solution $S_n$ in terms of subproblems $S_k$? Unfortunately, we cannot do that. Here is why. Consider the optimal solution if we can choose items 1 through 4 only.

Solution

Items chosen are 1, 2, 3, 4
Total weight: $2 + 3 + 4 + 5 = 14$
Total value: $3 + 4 + 5 + 8 = 20$

Now consider the optimal solution when items 1 through 5 are available

Ref: Handouts Page no. 92

Effect of calling max heap

Solution:
The smallest key is in the root in a min heap; in the max heap, the largest is in the root.

Ref: Handouts Page no. 40
Suggest the criteria for measuring algorithms. Also discuss the issues need to be discussed in the algorithm design.

Solutions:-

In order to design good algorithms, we must first agree the criteria for measuring algorithms. The emphasis in this course will be on the design of efficient algorithm, and hence we will measure algorithms in terms of the amount of computational resources that the algorithm requires. These resources include mostly running time and memory. Depending on the application, there may be other elements that are taken into account, such as the number disk accesses in a database program or the communication bandwidth in a networking application.

Ref: Handouts Page no. 9