2016170
Shubh Maheshwari

Assignment-3
Algorithm

1) i) The time complexity would change from $O(n)$ to $O(n+n\log n)$ or ~~$O(n\log n)$ bcc~~ $O(n\log n)$ because most of the elements would lie in one bucket.

ii) The memory utilized would change from $O(n)$ to $O(2^*n)$ or $O(n)$

iii) Based of which type of distribution given we might divide the range in different type. Hence instead of having n evenly spaced contains we could have unevenly spaced contains.

eg. n contains which get the range $\sum_{i=1}^{n} \frac{1}{n} \times \log i$; hence we can easily distribute them and apply bucket sort

2) One method for topo sort is by using a stack to store all nodes which have 0 incoming edges.

Hence if G = graph with V vertices and E edges

⇒ E = {{u1, v1}, {u2, v2} ... } (outgoing node, incoming node)

Algo:

~~for v in V:~~

~~for~~ e in E:

if e.v_i = v // the

~~for [e.v_i]~~

for e in E

cont [e.v_i] ++ // we store the count // of incoming edges for // each vertex

for v in V:

if count [v] == 0:

stack.push [v]

~~print (Stab~~

~~for~~ while (Stack not empty):

print (Stack.top())

v = stack.top()

for e in E:

~~sta~~ if e.u == v:

count [e.v] - -;

if (count [e.v] == 0)

stack.push [e.v]

Hence we remove the top node from the stack and check which nodes have 0 incoming edges in the new graph.

We repeat this until no node is left.

Hence complexity of this sort is O(V+E) no. of edges + no. of vertices

ii) Toposort is very important for many graph algorithm used to find shortest path, etc.

3) 

Algo :

$$\text{cost}(n) = \max\left(\sum_{i=1}^{n} (p_i \times \cos(n-i))\right)$$

Hence

i) $T(n) = \sum_{j=1}^{n} T(n-j) + n$

ii) $T(n) = 1 + T1 + T2 \quad \ldots T(n-1) + n$

$T(n+1) = T1 \ldots \quad T(n) + n+1$

$T(n+1) \geq T(n) + 1$

$-T(n)$

$T(n+1) = 2T(n) + 1$

1 can be ignored

$$T(n) = 2^n \times T(0)$$

$$= 2^n = O(2^n)$$

iii) Computing the minimum time complexity of Matrix chain multiplication is very similar to the rod cutting problem

4) 'N' fields are located in a circle and we need to maximize the total power:

```
b[size] = -1  // initialize everything to -1
b[1] = a[1]
b[0] = a[0]
powfunc(n,a,b)
{
    if b[n] != 0 :
        return b[n]

    if b[n] == -1
        return
    return max{ powfunc(n-1,a,b),
                a[n] + b[n-2]
                powfunc(n-2,a,b) }
}
```

5) The Algorithm used in this permutation is by swapping and going down the decision tree:

Algo:

permute(n, l, r):

for(int i=l; i<=r; i++)

{ if (a[i]==0 && location[i]

swap(a+

location[i] = i // position of each no.

val[i] = i value at each position

a[] = boolean array

permute(low, high)

check if the condition is satisfied.
set flag = true, else false

for(i=low; i<=high; i++)

{ swap(i, low);

permute(low+1, high);

swap(i, low);

}

if(low==high && flag==true)

{ print(array)

}