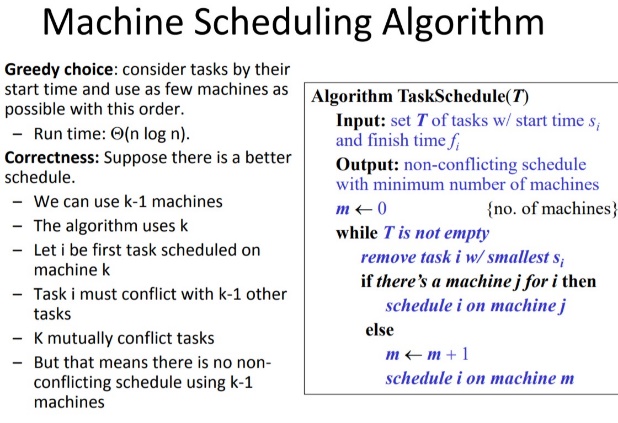
 6.Show log(n!) is O(nlogn)

Solution: log(n!)= log (n\*(n-1)\*(n-2)\*...2\*1)

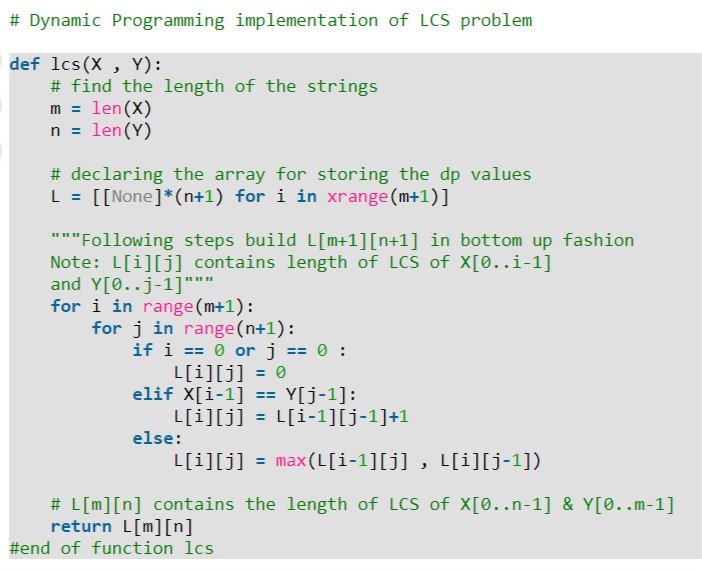
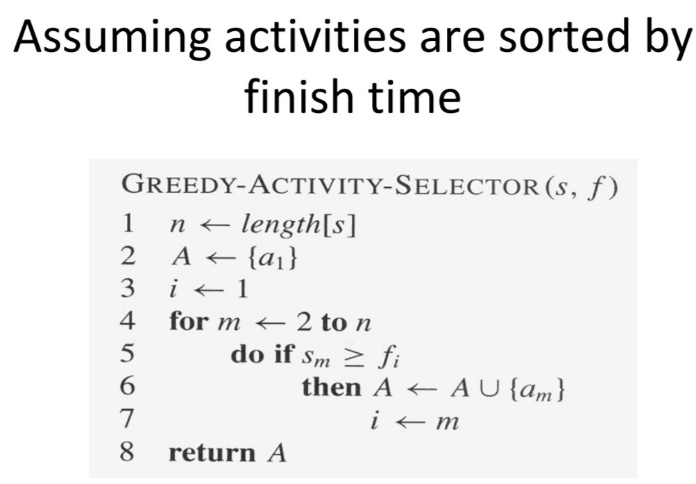
= logn + log(n-1) + log(n-2) + .. + log 2 + log1 < nlogn

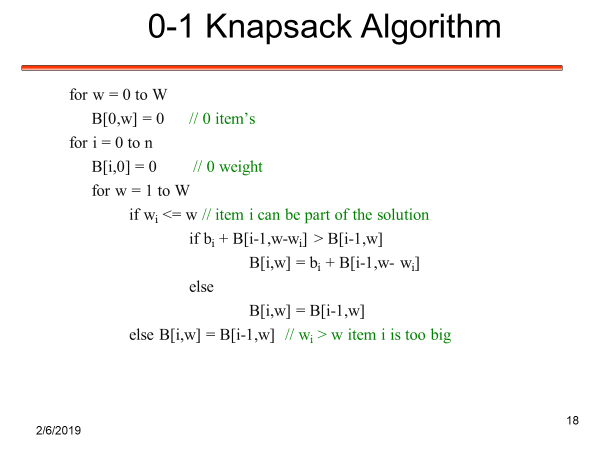
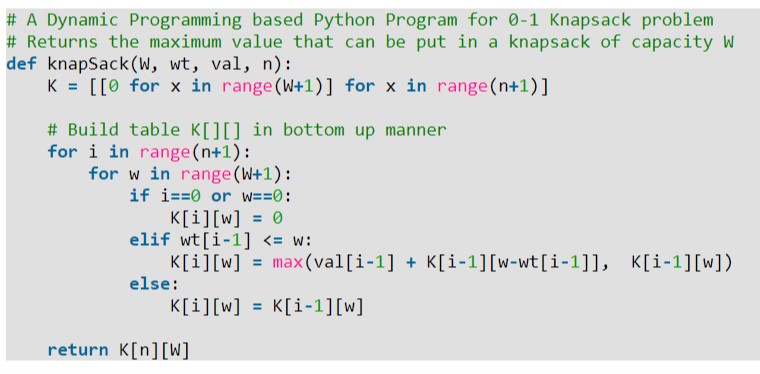
Therefore, O(nlogn)

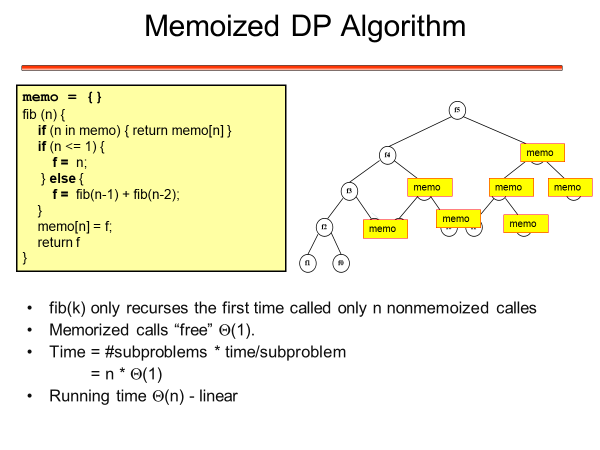
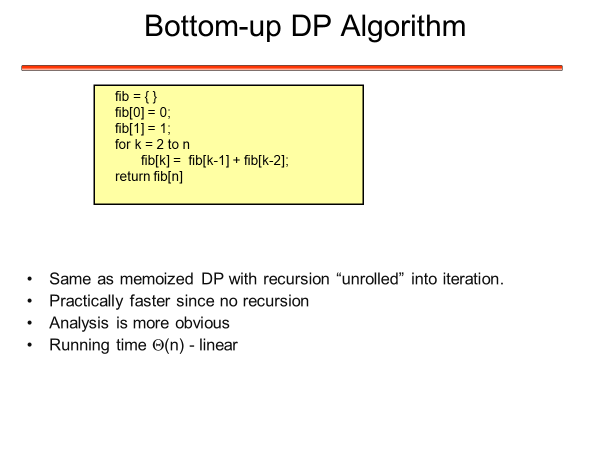


|  |  |  |
| --- | --- | --- |
| *RodCut(price):* | | |
|  | 1 | opt[0] = 0 |
|  | 2 | sol[0] = [] |
|  | 3 | **for** k = 1 **to** price.length |
|  | 4 | opt[k] = 0 |
|  | 5 | sol[k] = null |
|  | 6 | **for** i = 1 **to** k |
|  | 7 | **if** opt[k] < opt[k-i] + price[i] |
|  | 8 | opt[k] = opt[k-i] + price[i] |
|  | 9 | sol[k] = sol[k-i] + [i] |

|  |  |  |
| --- | --- | --- |
| *LIS(A):* | | |
|  | 1 | opt[0] = 0 |
|  | 2 | sol[0] = [] |
|  | 3 | **for** k = 1 **to** A.length |
|  | 4 | opt[k] = 1 |
|  | 5 | sol[k] = [A[k]] |
|  | 6 | **for** i = 1 **to** k-1 |
|  | 7 | **if** A[i] < A[k] **and** opt[k] < opt[i] + 1 |
|  | 8 | opt[k] = opt[i] + 1 |
|  | 9 | sol[k] = sol[i] + [A[i]] |
|  | 10 | best = 1 |
|  | 11 | **for** j = 2 **to** A.length |
|  | 12 | **if** opt[best] < opt[j] |
|  | 13 | best = j |







Input is array of unique characters along with their frequency of occurrences and output is Huffman Tree.

1. Create a leaf node for each unique character and build a min heap of all leaf nodes (Min Heap is used as a priority queue. The value of frequency field is used to compare two nodes in min heap. Initially, the least frequent character is at root)

2. Extract two nodes with the minimum frequency from the min heap.

3. Create a new internal node with frequency equal to the sum of the two nodes frequencies. Make the first extracted node as its left child and the other extracted node as its right child. Add this node to the min heap.

4. Repeat steps#2 and #3 until the heap contains only one node. The remaining node is the root node and the tree is complete.