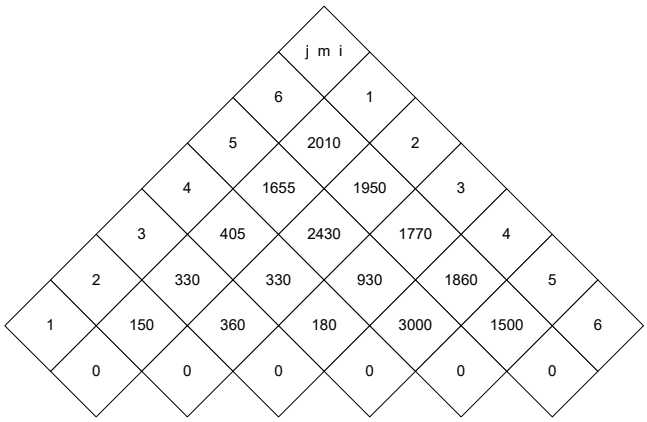
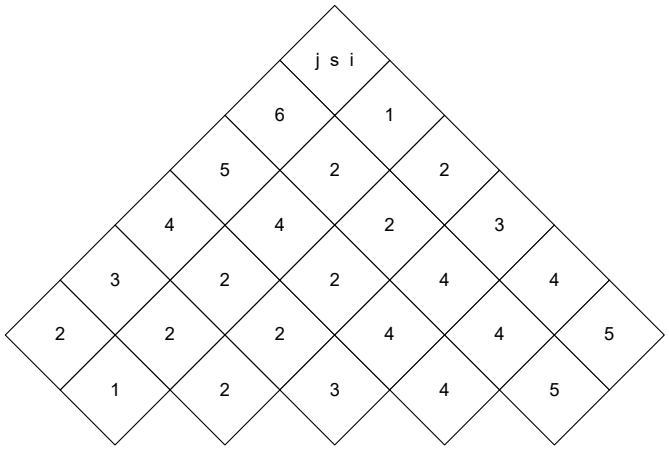
Assignment #5:

1. Exercise 15.2-1

   Solution:





2. Given an unsorted array $A[1, 2, ..., 2n]$ of numbers containing an even number of elements, and two players who take turns alternately at selecting one of the numbers at either end of the array (of remaining elements). Each wants to maximize the total of the numbers selected by them minus that of the other player, show how to use dynamic programming to obtain their best strategies.

   Solution:

   Since at each step the array gets shortened by one element either at the beginning or the end of the previous array, we will get to see as subproblems only subarrays consisting of consecutive elements – of the type

$A[i, i+1, ..., j]$. These are the subproblems and there are $\Theta(n^2)$ such problems. We will need the solution of smaller problems in order to solve larger problems and so we will process these in increasing order of $j - i$. It is Player I's turn to play whenever $j - i$ is odd and player II's turn when it is even. Payoff to Player I and Player II are negatives of each other. Here comes the recursion equation: Let $m[i, j]$ represent the payoff to the player who has the first move if the problem only consists of $A[i, i+1, ..., j]$. The:

$$\begin{aligned} m[i, i] &= A[i] \qquad \text{for all } i \\ m[i, j] &= \max\{A[i] - m[i+1, j], A[j] - m[i, j-1]\} \end{aligned}$$

Computational complexity: $\Theta(n^2)$. Recovery of the actual solution is done by keeping which of the two in the above equation produces the maximum.

3. Consider the activity selection problem discussed before but with profits. Activity i $(= 1, 2, ..., n)$ has three values associated with it. s_i the start time, f_i the finish time and p_i the profit. We want to select a subset of nonoverlapping activities whose total profit is maximum. Show how to use dynamic programming to solve this problem. (Hint: Read 16.1 in your book)

   Solution:

   Let $T[i] = \{j : s_j \geq s_i\} = \{i, i+1, ..., n\}$ **after sorting in increasing order** of s_i. Let $P[i]$ be the maximum profit from activities in $T[i]$. Let $k[i] = \min[j : s_j \geq f_i]$. Compute these as follows:

   (a) Sort the s_i values: Renumber the activities according to this sorted order. Complexity: $\Theta(n \lg n)$.

   (b) To find $k[i]$ for all i: Do binary search to find the right spot for f_i among the sorted list of s_j. Complexity: $\Theta(\lg n)$ per index i. Overall $\Theta(n \lg n)$

   (c) Derive a recursion equation connecting $P[i]$ to $P[j]; j \geq i+1$. There are two choices at the beginning of problem $P[i]$: include activity i or not.

   $$\begin{aligned} P[i] &= \max\{[p_i + P[k[i]], P[i+1]\}; i < n \\ P[n] &= p_n \end{aligned}$$

   These values are calculated in decreasing order of values of i. Complexity: To compute $P[i]$ for all i: $\Theta(n)$ – one addition and one comparison per value. Overall complexity: $\Theta(n \lg n)$.

4. Given a string $A[1, 2, ..., n]$ of numbers, find the subsequence $B[1, 2, ..., m]$ with $B[i] < B[i+1]$ for $i = 1, 2, ..., m-1$ such that the value of m is maximum.

   Solution:

Since you were prohibited from sorting and using the two sequences, here is a direct approach using dynamic programming:

Let $l[j]$ = the length of a longest increasing subsequence which ends in the element $A[j]$. Then we have:

$$\begin{aligned} l[1] &= 1 \\ l[j] &= \{1 + \max_{\substack{i: \\ i<j; A[i]<A[j]}} [l_i]\} \qquad j \geq 2 \end{aligned}$$

We keep track of the value of i that produces the maximum for each j to recover the solution. All this is done by the following pseudocode:

LIS(A)

1. $n \leftarrow length[A]$
2. **for** $j \leftarrow 1$ **to** n
3. **do** $l[j] \leftarrow 1$
4. $\pi[j] \leftarrow$NIL
5. **for** $i \leftarrow 1$ **to** $j-1$
6. **do if** $A[i] < A[j]$ and $1 + l[i] > l[j]$
7. **then** $l[j] \leftarrow 1 + l[i]$
8. $\pi[j] \leftarrow i$
9. **return** l and π

A simple implementation of this gives a $\Theta(n^2)$ algorithm. We can make it work in $\Theta(n \lg n)$ time by a better implementation as follows. For a fixed j and for a given value $v < A[j]$, it suffices to record the index $i < j$ for which $A[i] = v$ and $l[i]$ is the largest. Thus, if we have two indices i and k satisfying the relations $i < j; k < j; A[i] = A[k] < A[j]; l[k] \leq l[i]$ we need not consider the index k for computing $l[j]$. Let the set of triples over $A[1, 2, ..., j-1]$ be $(v_1, i_1, l_1), (v_2, i_2, l_2), ..., (v_p, i_p, l_p)$ where the triples are of the form (value, index, length).

- For the triple (v_k, i_k, l_k)

$$\begin{aligned} v_k &= A[i_k] \\ l_k &= l[i_k] \end{aligned}$$

- $v_1 < v_2 < v_3 < ... < v_p$
- $v_m < v_r; l_m \geq l_r \Longrightarrow$ we can discard the triple (v_r, i_r, l_r) from consideration for computing $l[j]$.
- Thus, after the above elimination, we have triples satisfying both the following relations: $v_1 < v_2 < v_3 < ... < v_p$ and $l_1 < l_2 < l_3 < ... < l_p$

- To evaluate

$$\max_{\substack{i: \\ i<j; A[i]<A[j]}} [l_i]$$

  over these type of indices, we can do a binary search taking $O(\lg n)$ steps since there are at most n indices to consider.
- Now this may eliminate some of our triplets for future consideration.
- This makes the algorithm take $\Theta(n \lg n)$ time overall.

There is another algorithm which also takes the same time but is based on the greedy approach to solve another problem that is related to this one.

5. Problem 15-6 (page 408)

   Solution:

   $t(x)$ = the best total value for subtree rooted at x(possibly including x).

   $u(x)$ = the best total value for subtree rooted at x (excluding x). Then

$$\begin{aligned} u(x) &= \sum_{i \in \{\text{children}(x)\}} t(i) \\ t(x) &= \max[u(x), r(x) + \sum_{i \in \{\text{children}(x)\}} u(i)] \end{aligned}$$

   Compute these values from the leaf nodes up. Boundary conditions:

$$\begin{aligned} u(x) &= 0 \text{ for leaf nodes} \\ t(x) &= \max[0, r(x)] \text{ for leaf nodes} \end{aligned}$$

   Complexity: $\Theta(n)$ since each node is visited twice and each edge twice.

6. 16.2-2: 0/1 Knapsack Problem:

   Solution:

   Solution:

   Given n items; i^{th} item has an integral weight w_i and a value v_i. The capacity of the knapsack is W. We want to solve the problem:

$$\begin{aligned} \sum_{i=1}^{n} w_i x_i &\leq W \\ & \max \sum_{i=1}^{n} v_i x_i \\ x_i &\in \{0, 1\}; i = 1, 2, ..., n \end{aligned}$$

Let $m[j,t]; \{j = 1,2,...,n; t \leq W; \text{integer}\}$ represent the maximum in the problem

$$\begin{aligned} \sum_{i=1}^{j} w_i x_i &\leq t \\ & \max \sum_{i=1}^{j} v_i x_i \\ x_i &\in \{0,1\}; i = 1,2,...,j \end{aligned}$$

where $t \leq W$ and is an integer. Then

$$m[1,t] = \begin{cases} 0 & t < w_1 \\ v_1 & t \geq w_1 \end{cases}$$

$$m[j,t] = \begin{cases} m[j-1,t] & t < w_j \\ \max\{m[j-1,t], v_j + m[j-1,t-w_j]\} & t \geq w_j \end{cases} \qquad n \geq j > 1$$

These values are computed in increasing order of values of j. For each set of values of j and t we need one addition and one comparison. Thus the overall complexity is $\Theta(nW)$.