Assignment #5:

1. Consider the activity selection problem discussed before but with profits. Activity $i (= 1, 2, \ldots, n)$ 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 $T[i] = \{j : s_j \geq s_i\} = \{i, i + 1, \ldots, n\}$ 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.

$$P[i] = \max\{p_i + P[k[i]], P[i + 1]\}; i < n$$

$$P[n] = p_n$$

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: $\Theta(n \lg n)$.

2. Exercise 15.4-5. (page 356) Give an $O(n^2)$ algorithm to find the longest monotonically increasing subsequence of a sequence of numbers.

**Solution:**

If we interpret "monotonically increasing" to mean nondecreasing, then we get an easy solution: Given $A[1, 2, \ldots, n]$ sort it to get another sequence $B[1, 2, \ldots, n]$. Now find a LCS of these two sequences. Sorting takes $\Theta(n \lg n)$ time. LCS takes $\Theta(n^2)$ time. So the overall complexity is $\Theta(n^2)$.

Now suppose equal elements are not allowed in the subsequence. So the problem becomes: Given a string $A[1, 2, \ldots, n]$ of numbers, find the subsequence $B[1, 2, \ldots, m]$ with $B[i] < B[i + 1]$ for $i = 1, 2, \ldots, m - 1$ such that the value of $m$ is maximum.

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:

\[
\begin{align*}
  l[1] &= 1 \\
  l[j] &= \{1 + \max_{i<j; A[i]<A[j]} [l_i]\} & j \geq 2
\end{align*}
\]

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 ← length[A]
2. for j ← 1 to n
3.   do l[j] ← 1
4.   π[j] ← NIL
5.   for i ← 1 to j - 1
7.       then $l[j] ← 1 + l[i]$
8.       π[j] ← i
9. return l and π
```

A simple implementation of this gives a $Θ(n^2)$ algorithm. We can make it work in $Θ(n \lg n)$ 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 $A[1, 2, \ldots, j - 1]$ be $(v_1, i_1, l_1), (v_2, i_2, l_2), \ldots, (v_p, i_p, l_p)$ where the triples are of the form (value, index, length).

- For the triple $(v_k, i_k, l_k)$
  \[
  v_k = A[i_k] \\
  l_k = l[i_k]
  \]

- $v_1 < v_2 < v_3 < \ldots < v_p$
- $v_m < v_r; l_m \geq l_r \implies$ 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: $v_1 < v_2 < v_3 < \ldots < v_p$ and $l_1 < l_2 < l_3 < \ldots < l_p$
• To evaluate
\[
\max_{i<j; A[i] < A[j]} \lfloor t_i \rfloor
\]
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.

3. 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
\[
\begin{align*}
  u(x) &= \sum_{i \in \{\text{children}(x)\}} t(i) \\
  t(x) &= \max[u(x), r(x) + \sum_{i \in \{\text{children}(x)\}} u(i)]
\end{align*}
\]

Compute these values from the leaf nodes up. Boundary conditions:
\[
\begin{align*}
  u(x) &= 0 \text{ for leaf nodes} \\
  t(x) &= \max[0, r(x)] \text{ for leaf nodes}
\end{align*}
\]

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

4. 16.2-2: 0/1 Knapsack Problem:

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:
\[
\sum_{i=1}^{n} w_i x_i \leq W
\]

\[
\max \sum_{i=1}^{n} v_i x_i
\]
\[
x_i \in \{0, 1\}; i = 1, 2, ..., n
\]
Let \( m[j, t]; \{ j = 1, 2, ..., n; t \leq W; \text{integer}\} \) represent the maximum in the problem

\[
\sum_{i=1}^{j} w_i x_i \leq t \\
\max \sum_{i=1}^{j} v_i x_i
\]

where \( t \leq W \) and is an integer. Then

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

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

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 \( \Theta(nW) \).

5. Consider the following problem faced by an advertising company. Billboards may be placed on a highway at any of \( n \) possible locations that are known to you along a highway. The highway is \( M \) miles long and the locations are at \( x_1 < x_2 < ... < x_n \) miles from one end of the highway. Regulations of the highway department prohibit any two billboards from being 5 miles or less apart. Placing a billboard at location \( i \) is worth \( r_i > 0 \) to the advertising agency. The input to the problem are the vectors \( x \) and \( r \) and the value of \( M \) and the separation 5. Give a dynamic programming algorithm to find the optimal set of locations to maximize total revenue.

**Solution:** Consider a sequential decision making process as follows: We either place a billboard at location at \( x_n \) or we do not. If we do not, the problem reduces in size by one to finding the best combination from the first \( n - 1 \) locations. For each location \( i \) let \( e(i) = \max\{j : x_j < x_i - 5\} \); if this set is empty, then \( e(i) = 0 \). These can be computed in \( O(n) \) time [How?] If we do have a billboard at location \( n \), then we get a revenue \( r_n \) from it but we are prohibited from having another billboard at locations \( x_i; i > e(n) \). Hence, we must solve the following subproblems: Find the maximum revenue with the same minimum separation constraint from locations \( x_1, x_2, ... x_j \). Let the maximum revenue for this be denoted by \( P_j \). The our recursion equation relating these will be by the above argument:

\[
\begin{align*}
P_0 &= 0; P_1 = r_1 \\
P_j &= \max[P_{j-1}, r_j + P_{e(j)}]; j \geq 2
\end{align*}
\]

Complexity of this is \( O(n) \).