1

Program set 3: solution

This set of questions focus on dynamic programming.

1. Knapsack without repetitions. Consider the following knapsack problem:

   The total weight limit $W = 10$ and

Item	Weight	Value
1	6	\$30
2	3	\$14
3	4	\$16
4	2	\$9

   Solve this problem using the dynamic programming algorithm presented in class. Please show the two dimensional table $L(w, j)$ for $w = 0, 1, \ldots, W$ and $j = 1, 2, 3, 4$.

Solution : Subproblems definition

Let $V(n, W)$ be the maximum value obtainable with item 1,2,...,n and weight W.

Recursive Formulation

To get $V(n, W)$ we consider

- Whether we can accommodate the item in the current weight limit
  $$w_i \le W$$
- Whether accommodating the current item will lead to a greater value for that weight.
  $$V(i, W) = V(i-1, W - w_i) + v_i$$

So our decision is reduced to whether or not to include item n or not. Our objective here is to maximize the value, hence we take the maximum of these values over all i and w_i.

$$V(n, W) = \max \begin{cases} V(n-1, W - w_n) + \boxed{v_n} & \boxed{\text{Use item n}} \\ V(n-1, W) & \boxed{\text{Discard item n}} \end{cases}$$

Base cases:

$$V(i, 0) = 0, for\ i = 0,1, \ldots, n$$
$$V(0, j) = 0, for\ j = 0,1, \ldots, W$$

Pseudo-code

KnapSack(v, w, n, W)
{
  for ($w = 0$ to W) $V[0, w] = 0$;
  for ($i = 1$ to n)
    for ($w = 0$ to W)
      if ($w[i] \le w$)
        $V[i, w] = \max\{V[i-1, w], v[i] + V[i-1, w - w[i]]\}$;
      else
        $V[i, w] = V[i-1, w]$;
  return $V[n, W]$;
}

put.

KNAPSACK PROBLEM : FIXED WEIGHT : DYNAMIC PROGRAMMING.

Problem :- Consider the following knapsack problem.

item	weight	value ($)
1	5	10
2.	4	40
3	6	30
4	3	50

4

what is the maximum value you can obtain if you can only put in a max. weight of 10. (W=10) (N=4)

1. Build an Item x Weight array which contains values.

~~$V[N][W] = [0]_{4\times10}$~~ $V[N+1][W+1] = [0]_{5\times11}$

$5-4=1$ $9-4=5$

V	weight 0	wt 1	wt 2	wt 3	wt 4	w5	w6	w7	w8	w9	w10
Item 0	0	0	0	0	0	0	0	0	0	0	0
item 2	0	0	0	0	0	40	40	40	40	40	40
item 2	0	0	0	0	40	40	.	.		40+10 = 50	50
item 3	0										30+40 = 70
item 4	0										

2. For 0th column, cap. of knapsack = 0, so, ~~exception~~ max value is 0.

~~$\forall[i]$~~ $V(i,0) = 0$, $i = 0$ to N

3. For 0th row, there are no items, so max value is 0.

4. Starting operation ~~rows~~-wise,

Row 1, Col 1 → ~~w~~ given ~~max~~ capacity = 1, can you accom. item 1 ?
in this case → NO.

Row 1, Col 2 → capacity = 2, ~~cat~~ can you accomodate item 1 ?
In this case → NO

⋮

Row 1, Col 5 → cap = 5, can you accomodate item 1 ?
$w_i \le W$ (currently 5) ? Yes. **Decision**
Fill $V[i,j]$ = Value of item 1 = 40

⋮

Row 1, Col 6 – Col 10 → We have only encountered item 1, ~~so~~ all values will be 40.

2

V	wt0	wt1	wt2	wt3	$\boxed{wt4}$	wt5	wt6	wt7	wt8	wt9	wt10
item0	0	0	0	0	0	0	0	0	0	0	0
item1	0	0	0	0	0	10	10	10	10	10	10
item2	0	0	0	0	40	40	40	40	40	50	50
item3	0										
item4	0										
~~item5~~											

5. Row 2, col 1 → cap = 1, can you accomodate item 2? NO

we check if $w_i \leq W$ [where W is the current wt / Col no]

⋮

Row 2, Col 4 → cap = 4, wt of item 2 = 4, can you accomodate? YES

$V[i,j] = w_i = 40$

Row 2, col 5 → cap = 5, we check
- Value $V[i,j]$ is greater without item i (2) ?
- Value $V[i,j]$ is lesser with item i (2) ?

To see this, we have to check, the previous row of same wt. ~~V[i-1,w]~~ V[i-1,

Here we decide, to use or not to use item i.

- Value with item 2 = 40
- Value without item 2 = 10

} max = 40 → inclued item 2. **Decision**

6. Now we check whether we can include both item 1 & 2.

In order to include both, the remaining wt $W - w_i \rightarrow W - w_2$ should be able to accomodate the previous item(s).

In this case, $(5) - (4) = 1 < w_{(2)-(1)} (w_1)$

Thus, we can't include both items. — **Decision**

⋮

7. Row 2, col 9 → cap = 9, ~~$w_2 \leq$~~ $w_2(4) \leq 9$ — we can include item 2.

→ is value greater with/without item 2 — with

→ can we include both — remaining wt $= 9 - 4 = 5 \geq w_i$ — both

$V[2,9] = 10 + 40 = 50$

```
Knapsack (v, w, n, W)
{
  for  w = 0 to W
       V[0,w] = 0        // base case

  for  i = 0 to n
       V[i, 0] = 0       // base case

  for  i = 1 to n
       for  j = 1 to W
            if ( w[i] ≤ j )
                V[i,j] = max { V[i-1, j] ,  V[i-1, j - w[i]] + v[i] }

            else
                V[i, j] = V[i-1, w]

  return V[n,W].
```

	[1] item	[2] wt	[3] value
[1]	1	3	10
[2]	2	1	5
[3]	3	2	20

$W = 4$

		[0] 0	[1] 1	[2] 2	[3] 3	[4] 4
[0]	item 0	0	0	0	0	0
[1]	1	0	.	.	.	.
[2]	2	0	.		.	
[3]	3	0				

$V[0,1] = V[0,2] = V[0,3] = V[0,4] = 0$

$V[1,0] = V[2,0] = V[3,0] = V[4,0] = 0$

$i=1, \quad j=1$

$w(1) \leq 1 \; ? \rightarrow 3 \leq 1 \; ?$ — NO

$V[1,1] = V[0,1] = 0$

$w(1) \leq 2 \; ? \rightarrow 3 \leq 2 \rightarrow$ NO

$V[1,2] = V[0,2] = 0$

$w(1) \leq 3 \; ? \rightarrow 3 \leq 3 \rightarrow$ Yes

$V[1,3] = \max\{V[0,3] \,,\, V[0,0] + v(1)\} = \max\{0, 0+10\} = 10$

$w(1) \leq 4 \; ? \rightarrow 3 \leq 4 \rightarrow$ yes

$V[1,4] = \max\{V[0,4] \,,\, V[0,4-3] + v(1)\} = \max\{0, 0+10\} = 10$

$i=2, \; j=1$

$w(2) \leq 1 \; ? \; 1 \leq 1 \rightarrow$ yes , $V[2,1] = \max\{V[1,1] \,,\, V[1,0] + v(2)\} = \max\{0, 0+5\} = 5$

$w(2) \leq 3 \; ? \rightarrow 1 \leq 3 ?$ yes

$V[2,3] = \max\{V[1,3] \,,\, V[1,2] + v(2)$

$= \max[10, 0+5]$

$= 10$

$w(2) \leq 4 \rightarrow$ yes

$V[2,4] = \max\{V[1,4] \,,\, v(2) + V[1,3]\}$

$= \max\{10, 5+10\}$

$= 15$

2. Give a dynamic programming algorithm for the following task.

Input: A list of n positive integers $a_1, a_2, \ldots, a_n$ and a number t.
Question: Does some subset of the a_i's add up to t? (You can use each a_i at most once.)

2

Solution: Subproblem definition and Recursive formulation

The running time should be $O(nt)$.

Consider the subproblem $L(i, s)$, which returns the answer of "Does a subset of $a_1, \ldots, a_i$ sum up to s?". Below are some substeps that will help you develop your algorithm.

a What are the two options we have regarding item i toward answering the subproblem $L(i, s)$?
The two options are using a_i or not using a_i.

b For each of the option, how would it change the subproblem? More specifically, what happens to the target sum and what happens to the set of available integers?
If we use a_i, the target sum will become $s - a_i$, and the available integers that can be used to achieve this target sum will become $a_1, \ldots, a_{i-1}$, which is captured by subproblem $L(i-1, s-a_i)$.
If we do not use a_i, the target sum will remain to be s and the available integers that can be used to achieve this target sum will become $a_1, \ldots, a_{i-1}$, which is captured by subproblem $L(i-1, s)$.

c Based on the answers to the previous two questions, write a recursive formula that expresses $L(i, s)$ using the solutions to smaller subproblems.
The recursive formula is $L(i, s) = L(i-1, s)$ OR $L(i-1, s-a_i)$. Here we assume that the subproblem returns either 0 or 1, where 0 means the answer is no, and 1 means that the answer is yes.

Pseudo-code

d Provide pseudocode for the dynamic programming algorithm that builds the solution table $L(i, s)$ and returns the correct answer to the final problem.

$L(i, 0) = True$ for $i = 1, \cdots, n$
$L(0, s) = False$ for $s = 1, \cdots, t$
for $i = 1, \cdots, n$
    for $s = 1, \cdots, t$
        if $s < a_i$: $L(i, s) = L(i-1, s)$
        else: $L(i, s) = L(i-1, s)$ OR $L(i-1, s-a_i)$
    end for
end for
return $L(n, t)$

e Modify the pseudocode such that it will not only return the correct "yes", "no" answer, but also return the exact subset if the answer is "yes".

$L(i, 0) = True$ for $i = 1, \cdots, n$
$S(i, 0) = \emptyset$ for $i = 1, \cdots, n$
$L(0, s) = False$ for $s = 1, \cdots, t$
$S(0, s) = \emptyset$ for $i = 1, \cdots, n$
for $i = 1, \cdots, n$
    for $s = 1, \cdots, t$
        if $s < a_i$: $L(i, s) = L(i-1, s)$; $S(i, s) = S(i-1, s)$
        else if $L(i-1, s)$: $L(i, s) = True$; $S(i, s) = S(i-1, s)$
        else if $L(i-1, s-a_i)$: $L(i, s) = True$; $S(i, s) = S(i-1, s-a_i) \cup i$
        else: $L(i, s) = False$; $S(i, s) = \emptyset$
    end for
end for
return $L(n, t)$ and $S(n, t)$

Complexity

- Table L has $(n+1) \times (t+1)$ entries
- Overall complexity is $O(nt)$

Sample code

http://www.geeksforgeeks.org/dynamic-programming-subset-sum-problem/

SUBSET SUM PROBLEM : WITHOUT REPLACEMENT

Input → $(a_1, a_2, \dots, a_n)$ & t

Does some subset of a_i's add upto t ?

~~[Set] = A Set = {3, 34, 4, 12, 5, 2} t = 9~~

Let $L(i, s)$ return the subset.

Solution

1. Given sum $= t$, can we include element i in the subset?

→ If we do include, the remaining sum $(s - a_i)$
i.e we can henceforth only include elements which sum upto $(s - a_i)$

Since we have already include a_i, we only choose from $a_1, \dots, a_{i-1}$

Subproblem: $L(i-1, s-a_i)$

→ If we don't include, the remaining sum (t)
we choose from $a_1, \dots, a_{i-1}$

2. Base case 1 : $L(i, 0) = 1$, $i = 0 \dots n$

Since sum $= 0$, we can't include any elements. Hence we can always find a NULL subset.

Base case 2 : $L(0, s) = 0$, $s = 1, \dots, t$

If sum > 0, & you have no items, it returns false.

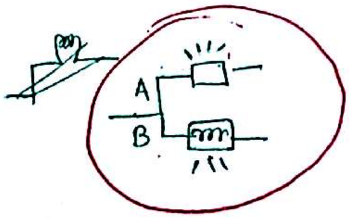
3. when $s < a_i$ → we can't include element, — Decision

$$L(i, s) = L(i-1, s)$$

4. If $a_i < s$, we can include — Decision

$$L(i, s) = L(i-1, s) \,||\, L(i-1, s-a_i)$$

A	B	A‖B
0	0	0
0	1	1
1	0	1
1	1	1



```
Subset-sum (A,t)
for i=1:n
    L(i,0) = 1 ,   S(i,0) = NULL
for s = 1:t
    L(0,s) = 0 ,   S(0,s) = NULL

for i = 1 to n
    for s = 1 to t
        if (s < ai)
            L(i,s) = L(i-1,s)
            S(i,s) = S(i-1,s)
        else
            L(i,s) = L(i-1,s) or L(i-1,s-ai)

            if (L(i,s) = L(i-1,s))
                S(i,s) = S(i-1,s)
            if (L(i,s) = L(i-1,s-ai))
                S(i,s) = S(i-1,s-ai) || i

    return L(n,t) & S(n,t)
```

$\{2, 3, 7, 8, 10\}$ $t = 11$

	0	1	2	3	4	5	6	7	8	9	10	11
2	1	0	1	0	0	0	0	0	0	0	0	0
3	1	0	1	1	0	1	0	0	0	0	0	0
7	1	0	1	1	0	1	0	1	0	1	1	0
8	1	0	1	1	0	1	0	1	1	1	1	1
10	1	0	1	1	0	1	0	1	1	1	1	1

3 ← not coming → 3

8 ← not coming from here → 8

prev row same wt

$L(i,j) = L(i-1, j) \,||\, L(i-1, s-a_i)$

step 1 → can we make s with $i-1$? (if prev row 1, it will be 1) (3,2)

step 2 → can we include i?

step 3 → can we make s with i? (go ~~seven~~ a_i steps back on prev row, if it is 1 then 1)

Scanned by CamScanner

If we start looking for i^* at $j = i - 1$ and increment j untill we find the correct value of i^*, the overall complexity can be reduced to $O(n)$.

3

5.

6.8. **Given two strings $x = x_1x_2\cdots x_n$ and $y = y_1y_2\cdots y_m$, we wish to find the length of their *longest common substring*, that is, the largest k for which there are indices i and j with $x_ix_{i+1}\cdots x_{i+k-1} = y_jy_{j+1}\cdots y_{j+k-1}$. Show how to do this in time $O(mn)$.**

Solution: Subproblem Definition

For $1 \le i \le n \; and \; 1 \le j \le m$, we define a subproblem $L(i,j)$ to be the length of the longest common substring of x and y terminating at $x_i \; and \; y_j$.

Recursion Formulation

- If $x_i \ne y_j$, the character is not common in the two strings and $L(i,j) = 0$.
- If $x_i = y_j$, the character is common and the length of the common substring is $1 + L(i-1, j-1)$.

Overall recursive formulation:

$$L(i,j) = \begin{cases} L(i-1, j-1) + 1 & \text{if } \mathrm{equal}(x_i, y_j) = 1 \\ 0 & \text{otherwise} \end{cases}$$

Base case:
For all $1 \le i \le n \; and \; 1 \le j \le m,$
$L(0,0) = 0$
$L(i,0) = 0$
$L(0,j) = 0$

We are interested in the maximum value of $L(i,j)$ over all $1 \le i \le n \; and \; 1 \le j \le m$.

Pseudo-code

$for \; i = 0 \; to \; n$
    $L(i,0) = 0$
$for \; j = 0 \; to \; m$
    $L(0,j) = 0$ //base cases

$for \; i = 1 \; to \; n$
    $for \; j = 1 \; to \; m$
        $if \; (x_i = y_j)$
            $L(i,j) = 1 + L(i-1, j-1)$
        $else$
            $L(i,j) = 0$
$return$ max $\{ L(i,j) \}$

Complexity

- We have $m * n$ subproblems and each takes constant time to evaluate through the recursion
- Overall complexity $O(mn)$.

Sample code
http://algorithms.tutorialhorizon.com/dynamic-programming-longest-common-substring/

LONGEST COMMON SUBSTRING

① Subproblem: $L(i,j)$ be the length of the longest common substring for $1 \le i \le n$ & $1 \le j \le m$ terminatinating at x_i & y_j

② Recursion: if $x_i \neq y_j$, char. is not common, $L(i,j) = 0$

if $x_i = y_j$, char. is common $L(i,j) = 1 + L(i-1, j-1)$.

③ We are interestes in the longest subsequence

$$L(i,j) = \begin{cases} 1 + L(i-1, j-1) & , \text{ if } x_i = y_j \\ 0 & , \text{ otherwise.} \end{cases}$$

④ Base cases:

$L(0,0) = 0$, ~~$\forall i, j$~~, if both strings are null, no comom substring

$L(i,0) = 0 \ \forall i$, ~~&~~ $L(0,j) = 0 \ \forall j$ } any ~~subs~~ string is null, no common substring.

⑤
```
for i = 1 to n
 { for j = 1 to m
   {
     if (xi = yj)
       { L(i, j) = 1 + L(i-1, j-1)
       }
     else
       { L(i,j) = 0
       }
   }
 }
return max L(i,j)
```

~~x = [AND]~~ ~~y = [CANADA]~~ , n = 3, m = 6

x = [AND] y = [CANADA]

i=1, j=1

x(1) = y(1)? → no L(1,1) = 0

i=1, j=2

x(1) = y(2)? → yes L(1,2) = 1 + L(0,1) = 1

i=1, j=3

x(1) = y(3) = ? → no L(1,3) = 0

i=1, j=4

x(1) = y(4)? → yes L(1,4) = 1 + L(0,3)

	0	C	A	N	A	D	A
0	0	0	0	0	0	0	0
A	0	0	1	0	1	0	1
N	0	0	0	$\boxed{2}$	0	0	0
D	0	0	0	0	0	1	0

---

i=2, j=1 → no → L(2,1) = 0

:

i=2, j=3 → yes → L(2,3) = 1 + L(1,2) = 1+1 = 2

i=2, j=3 →

i=3, j=1 → no → 0 (3,1)

j=2 → no → 0 (3,2)

:

i=3, j=5 → yes L(3,5) = 1 + L(2,4) = 1 + 0 = 1

4.

4

6.3. Yuckdonald's is considering opening a series of restaurants along Quaint Valley Highway (QVH). The n possible locations are along a straight line, and the distances of these locations from the start of QVH are, in miles and in increasing order, $m_1, m_2, \ldots, m_n$. The constraints are as follows:

- At each location, Yuckdonald's may open at most one restaurant. The expected profit from opening a restaurant at location i is p_i, where $p_i > 0$ and $i = 1, 2, \ldots, n$.
- Any two restaurants should be at least k miles apart, where k is a positive integer.

Give an efficient algorithm to compute the maximum expected total profit subject to the given constraints.

Solution: Subproblems definition

Let $D(i)$ be the maximum profit Yuckdonald's can obtain from locations 1 *to* i.

Recursive formulation

We have two choices for location i

- Not to open a restaurant at location i, in which case the optimal profit can be obtained from location 1,2,...,$(i-1)$ by $D(i-1)$.
- Open a restaurant at location i. In this case there is an expected profit of p_i. After building at location i, the nearest location to the left where we can build is

$$i^* = \max\{j \le i : m_j \le m_i - k\}$$

The profit of this can be denoted by $D(i^*)$.

Since we want to maximize our profit, we are interested in the maximum of these values over all i.

$$D(i) = \max\{D(i-1), p_i + D(i^*)\}$$

Base case: $D(0) = 0$.

Pseudo-code

Assuming i^* are known

$D(0) = 0$ //base case

$for\ i = 1\ to\ n$

  $noRestaurant = D(i-1)$

  $Restaurant = p_i + D(i^*)$ //storing both estimated profits in variables

  $if\ (noRestaurant > Restaurant)$ //deciding if profit is maximum with or without i

    $D(i) = noRestautant$

  $else$

    $D(i) = Restaurant$

$return\ D(n)$

Complexity

- We have n subproblems
- Each subproblem requires finding an i^* which can be done in time $O(logn)$ by using binary search on the ordered list of location
- And computing the maximum of two values which can be done in constant time
- Overall complexity is $O(nlogn)$

Sample code:

http://stackoverflow.com/questions/35673228/restaurant-maximum-profit-using-dynamic-programming

****Improving the complexity****

i^* can be computed in $O(n)$ time by using the fact that it is an ordered list i.e.

$0 = 1^* \le 2^* \le \cdots \le n^* < n$.

YUCKDONALD'S RESTAURANT : MAXIMUM PROFIT

Subproblem : $D(i)$ is the max profit from loc. 0 to i

Recursion :

1. We will not open a rest. at i →

   when optimum profit obtained from 1, 2, …, i-1

   $D(i-1)$

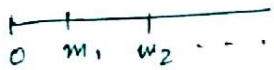
2. We will ~~include~~ open a rest at i →

   when optimum profit is obtain from rest at i & ~~optimum position from~~ optimum profit from nearest location to i.

   $p(i)$ — expected profit from loc. i

   $D(i^*)$ — expected profit from nearest location to i where we can build.

   $$i^* = \max\{\, j \le i : m_j \le m_i - k \,\}$$



3. We are interested in the maximum profit

   $$D(i) = \max\{\, D(i-1),\; p(i) + D(i^*) \,\}$$

4. Base case

   $$D(0) = 0$$

Scanned by CamScanner

Max profit (i, m, p)

{

D(0) = 0 // base case, profit = 0 , R(0) = 0 // if there is no rest at 0

for i = 1 to n

not_to_open_at_i = D(i-1)

open_at_i = p(i) + D(i^k)

if (not_to_open_at_i > open_at_i)

D(i) = not_to_open_at_i

R(i) = 0.

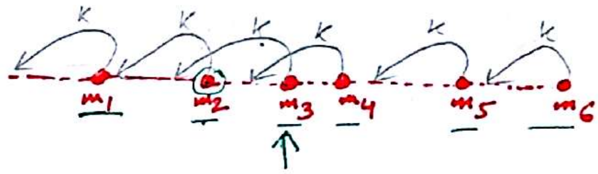
else

D(i) = open_at_i

R(i) = 1

return D(n) & R.

}



Start with profit = 0

→ we check at i, if not opening gives more profit than opening

→ if not opening is more, profit → Profit = Profit till i-1

→ if opening is more, profit of that + profit of nearest loc. to i ($m_j \leq m_i - k$)

Improving the i^* calculation

We can calculate i^* by binary search which takes $(\log n)$ time

$$i^* = \max\{j : m_j \leq m_i - k\}$$

But we can start looking for i^* at $j = i-1$ & increment j until we find the correct i^*.

```
Compute i*(m, k)
{   i*(1) = 0
    for i = 2 to n :
    {    j = i*(i-1)
         while (m_{j+1} <= m_i - k)
    →    {
              j = j+1
         }
         i*(i) = j
    }
    return i*[...]
}
```

$\boxed{O(n)}$

Reporting the location

```
Report (R, i*)
{
    j = n, S = NULL
    while j >= 1
    {  if (R[j] = 1)
          {  Insert m_j to S
             j = i*
          }
       else
          {  j = j-1
          }
    }
    return S
}
```

3.

6.2. You are going on a long trip. You start on the road at mile post 0. Along the way there are n hotels, at mile posts $a_1 < a_2 < \cdots < a_n$, where each a_i is measured from the starting point. The only places you are allowed to stop are at these hotels, but you can choose which of the hotels you stop at. You must stop at the final hotel (at distance a_n), which is your destination.

5

You'd ideally like to travel 200 miles a day, but this may not be possible (depending on the spacing of the hotels). If you travel x miles during a day, the *penalty* for that day is $(200 - x)^2$. You want to plan your trip so as to minimize the total penalty—that is, the sum, over all travel days, of the daily penalties.

Give an efficient algorithm that determines the optimal sequence of hotels at which to stop.

Solution: Suproblems definition

Let $OPT(i)$ be the minimum total penalty to get to hotel i.

Recursive formulation

To get $OPT(i)$, we consider all possible hotels j we can stay at the night before reaching hotel i. For each of these possibilities, the minimum penalty to reach i is the sum of:

- the minimum penalty $OPT(j)$ to reach j,
- and the cost $(200 - (a_j - a_i))^2$ of a one-day trip from j to i.

Because we are interested in the minimum penalty to reach i, we take the minimum of these values over all the j:

$$OPT(i) = \min_{0 \le j < i} \{OPT(j) + (200 - (a_j - a_i))^2\}$$

And the base case is $OPT(0) = 0$.

$OPT(k) + (200 - (a_k - a_j))^2$

Pseudo-code

```
// base case
OPT[0] = 0
// main loop
for i = 1...n:
  OPT[i] = min([OPT[j] +(200 − (a_j − a_i))^2 for j=0...i-1])
// final result
return OPT[n]
```

Complexity

- We have n subproblems,
- each subproblems i takes time O(i)

The overall complexity is

$$\sum_{i=1}^{n} O(i) = O(\sum_{i=1}^{n})i = O\frac{n(n-1)}{2} = O(n^2)$$

Sample code
http://www.solvemyproblem.net/Webed/forum/thread.asp?tid=1122

HOTEL PROBLEM

① Subproblem : OPT(i) is the minimum penalty to get to hotel i

② To get to i, we have to consider all j hotels that we have
Recursion to stay at night

→ OPT(j) is minimum penalty to get to j

→ ~~cost~~ penalty for one-day trip from (i) to (j)

$$(200 - (a_i - a_j))^2 \qquad (200 - x)^2$$

③ We are looking for min. penalty.

$$OPT(i) = \min_{\boxed{0 \le j < i}} \{ OPT(j) + (200 - (a_i - a_j))^2 \}$$

④ OPT(0) = 0 , since we are not going anywhere!!

⑤ for i = 1 to n .

$$OPT(i) = \min_{\cancel{0 \le j < i}} \{ OPT(j) + (200 - (a_i - a_j))^2 \} \text{ for } j = 0 \cdots (i-1).$$

n

```
best Route (int a[])
{
   S[] , R[]        // S - min penalty , R - prev stop with optimum cost
   S[0] = 0   R[0] = 0

   for (i = 1 to n)
   {  prev. sto = 0 , minpen = ∞,
      for (j = 0 to i-1)
      {
         if (S[j] + (200 - (a[i] - a[j]))^2 < min Penalty)
         {
               min pen = S[j] + (200 - (a[i] - a[j]))^2
               prev. stop = j
         }
      }
      S[i] = min pen.
      R[i] = prev. stop.
   }
} return S(n)
```

1 2 3 . . . n

Starting penalty = ∞

1. finds the min cost ~~S[i]~~ (min pen) & stores the stop which yeilded it (prev. stop)
2. it iterates over all possible prev. stops $0 \le j < i$ & stores the optimum post & cost for that i in S[i] & R[i]
3. It iterates for every hotel & cretes arrays S & R (1 x n) which stores the optimum stops for each hotel.

LONGEST PALINDROMIC SUBSEQUENCE

6.7. A subsequence is *palindromic* if it is the same whether read left to right or right to left. For instance, the sequence

$$A, C, G, T, G, T, C, A, A, A, A, T, C, G$$

has many palindromic subsequences, including A, C, G, C, A and A, A, A, A (on the other hand, the subsequence A, C, T is *not* palindromic). Devise an algorithm that takes a sequence $x[1 \ldots n]$ and returns the (length of the) longest palindromic subsequence. Its running time should be $O(n^2)$.

SOLUTION:

6.7. *Subproblems:* Define variables $L(i, j)$ for all $1 \leq i \leq j \leq n$ so that, in the course of the algorithm, each $L(i, j)$ is assigned the length of the longest palindromic subsequence of string $x[i, \cdots, j]$.

Algorithm and Recursion: The recursion will then be:

$$L(i, j) = \max\{L(i+1, j), L(i, j-1), L(i+1, j-1) + \text{equal}(x_i, x_j)\}$$

where equal(a, b) is 1 if a and b are the same character and is 0 otherwise, The initialization is the following:

$$\begin{array}{rr} \forall i, 1 \leq i \leq n \ , & L(i,i) = 0 \\ \forall i, 1 \leq i \leq n-1 \ , & L(i, i+1) = \text{equal}(x_i, x_i + 1) \end{array}$$

Correctness and Running Time: Consider the longest palindromic subsequence s of $x[i, \cdots, j]$ and focus on the elements x_i and x_j. There are then three possible cases:

- If both x_i and x_j are in s then they must be equal and $L(i, j) = L(i+1, j-1) + \text{equal}(x_i, x_j)$
- If x_i is not a part of s, then $L(i, j) = L(i+1, j)$.
- If x_j is not a part of s, then $L(i, j) = L(i, j-1)$.

Hence, the recursion handles all possible cases correctly. The running time of this algorithm is $O(n^2)$, as there are $O(n^2)$ subproblems and each takes $O(1)$ time to evaluate according to our recursion.

NOTE:

During the session, you pointed out that the algorithm doesn't work for the case when the length of the palindrome is even.(eg, abba or abhba). It is indeed true, there is an edge case to this.

The recursive formula is reduced to:

1. Every single character is a palindrome of length 1
   L(i, i) = 1 for all indexes i in given sequence

2. IF first and last characters are not same
   If (X[i] not equals X[j]) L(i, j) = max{L(i + 1, j),L(i, j - 1)}

3. If there are only 2 characters and both are same
   Else if (j == i + 1) L(i, j) = 2

4. If there are more than two characters, and first and last characters are same
   Else L(i, j) = L(i + 1, j - 1) + 2

In order to avoid the case where the algorithm cannot handle an even length palindrome, try a different approach.

Instead of iterating over every i and j, we consider the length of the substring. (say cl)

Say our string is 'abgba'.

so when cl =1, we only consider substring of length=1. Like---> a,b,g,b,a

when cl =2, we consider substrings of length =2 . Like ---> ab,bg,gb,ba

when cl = 3 e consider substrings of length 3. Like ---> abg,bgb,gba.

And so on .(for every length we keep increasing the starting point from which we consider the substring until every combination is possible).

and we construct a for loop like this:

```
for (cl=2; cl<=n; cl++)
  {
    for (i=0; i<n-cl+1; i++)
    {
      j = i+cl-1;
      if (str[i] == str[j] && cl == 2)
      L[i][j] = 2;
      else if (str[i] == str[j])
      L[i][j] = L[i+1][j-1] + 2;
      else
      L[i][j] = max(L[i][j-1], L[i+1][j]);
    }
  }
```

This handles both even and odd length palindromes.

You can find the full code here:
http://code.geeksforgeeks.org/index.php