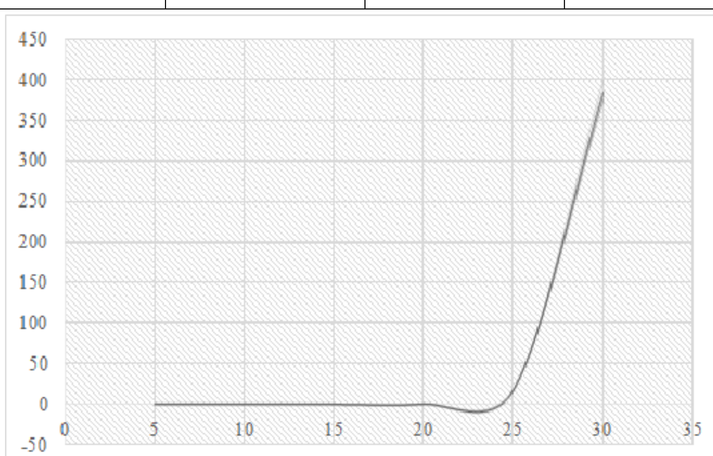
1 Recursive Algorithm and Analysis

To calculate a single value we make 4 recursive calls. We call *T(i+2,j)*, *T(I,j-2)*, and two calls to *T(i+1, j-1)*. This adds to 4 calls per *T(i,j)* plus base cases gives us 86 calls. To check this I added a print statement in the code set to print every time the function was accessed. It printed 86 times. Note that *val* is a global variable storing the "tree" as an array.

```
#----------------Implementation of Recursive ---------------------------
def recurit(i,j):
    #-------------Base Case---------
    #print("Called")
    if i == j:
        result = val[i]
        return result
    if j == i+1:
        result = max(val[i],val[j])
        return result
    #-----Recursive Algorithim----
    result = max(val[i] + min(recurit(i+2,j), recurit(i+1,j-1)), val[j] + min(recurit(i+1,j-1),recurit(i
        ,j-2)))
    print(result)
```

I tested the algorithim for a few different sizes of n using the Python time function. Before the algorithim *start = time.time()* is placed and after *end = time.time()* is placed. The table below is the values of *end - start* for values of n.

Size(n)	**Try 1**	**Try 2**	**Try 3**	**Average**
5	0.0004	0	0	0.0001
10	0.003	0.002	0.001	0.002
15	0.021	0.0245	0.0221	0.0226
20	0.334	0.334	0.331	0.333
25	17.279	17.269	18.694	17.747
30	374.512	398.508	379.910	384.175



It is difficult to test much further beyond n = *10* because it begins to take an unreasonable amount of time. Given the extreme run times we get to very quickly and the trend of the plot, performance is $0(2^n)$.

2 Dynamic Program Analysis

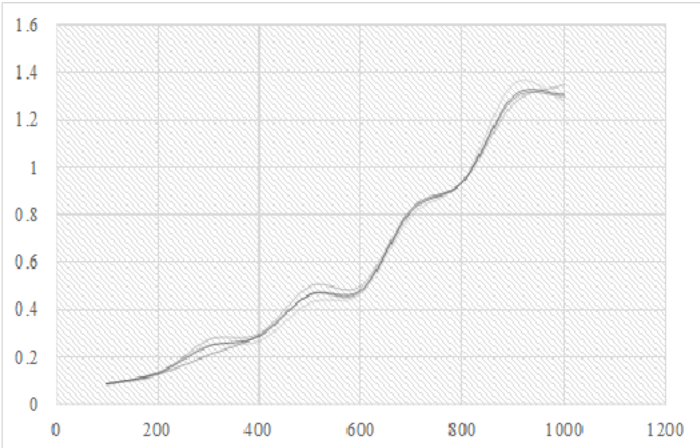
What follows is my implementation of dynamic programming that uses a table, *T*, to store partial results. Table *TB* is a table of the same size as *T* that used used for trace-back. Logic for filling both tables is done independently although they use the same algorithm so it could be rewritten to reflect this but I left it explicit. Later, beginning at the *while loop*, I walk back through the trace-back table, *TB*, and use its information fill *sel* which is the selections made in order.

```
#------------------Implementation of Dynamic Programming ------------------
def timber(n,val):
#------- Build the table ----------------------
    T = [[0 for x in range(n)] for y in range(n)]
    TB = [[0 for x in range(n)] for y in range(n)]

#------ Fill Table with base cases--------------
    for i in range(n-1,-1,-1):
        for j in range(n):
            if j >= i:
                if j == i:
                    T[i][j] = val[i]
                if j == i+1:
                    T[i][j] = max(val[i],val[j])
#---------Calculte remaining items -------------
    sel = []
    for i in range(n,-1,-1):
        for j in range(n):
            if j >= i and T[i][j] == 0:

                T[i][j] = max(val[i] + min(T[i+2][j], T[i+1][j-1]), val[j] + min(T[i+1][j-1],T[i][j-2]))

                #------ Logic For Filling Traceback Table ----------
                ii = val[i] + T[i+2][j]
                ij = val[i] + T[i+1][j-1]
                ji = val[i] + T[i+1][j-1]
                jj = val[j]+ T[i][j-2]

                if min(ii,ij)>min(jj,ji):
                    if ij<ii:
                        TB[i][j] = "lr"
                    else:
                        TB[i][j] = "ll"
                else:
                    if ji<jj:
                        TB[i][j] = "rl"
                    else:
                        TB[i][j] = "rr"
    x = 0
    y = n - 1
#----------Unravel Traceback Table --------------
    while i != j:
        if TB[x][y] == "ll":
            sel.append(i+1)
            i = i+1
            sel.append(i+1)
```

```
            i = i+1
        if TB[x][y]:
            sel.append(i+1)
            i = 1 +1
            sel.append(j+1)
            j = j - 1
        if TB[x][y] == ("jj"):
            sel.append(j+1)
            j = j - 1
            sel.append(j+1)
            j = j-1
        if TB[x][y] == ("ji"):
            sel.append(j+1)
            j = j - 1
            sel.append(i+1)
            i = i+1
        if TB[x][y] == ("i"):
            sel.append(i+1)
            i = i+1
        if TB[x][y] == ("j"):
            sel.append(j+1)
            j = j-1
    print(T[0][n-1])
    print(sel)
    print(result)
```

Testing this algorithm was done in the same method as the recursive algorithm implementation. The sample size for n varies significantly as performance is improved greatly so much larger values are capable. There is also a larger variance between n than the recursive implementation to better observe behavior.

Size(n)	**Try 1**	**Try 2**	**Try 3**	**Average**
100	0.089165003	0.082165003	0.084995003	0.085441669
200	0.12883143	0.138142958	0.123142958	0.130039115
300	0.210216041	0.25160408	0.270916041	0.244245387
400	0.29426982	0.268698205	0.29698205	0.286650025
500	0.464083672	0.430408367	0.499083672	0.464525237
600	0.478330354	0.471068804	0.500803544	0.483400901
700	0.82800746	0.798300746	0.82829348	0.818200562
800	0.940562963	0.942930563	0.939856296	0.941116608
900	1.28851801	1.351801014	1.27101801	1.303779011
1000	1.299914837	1.277948369	1.351914837	1.309926014



Storing partial results improved performance greatly while given the same results. Even though this projects only asks for "a few values of n" we could have continued much further until run time becomes unrealistic. Compared to the behavior of the recursive algorithm and considering the table and plot we can estimate a worst-case run-time of $O(n^2)$.

3 Validation

See above code for implementation of trace back in the Dynamic Programming algorithm. What follows is validation on the given input in the project description:

$$Input : [33, 28, 35, 23, 23, 25, 37, 40, 42, 24, 38, 29, 22, 40, 36, 42, 39, 37, 45, 32]$$

$$n = 20$$

350
[1, 2, 3, 20, 4, 5, 19, 6, 7, 18, 8, 9, 10, 11, 17, 12, 13, 16, 14, 15]

4 Appendix

This contains my code in its entirety. If unwanted please ignore.

```
from ast import While
from ctypes import sizeof
from math import ceil
import random
from turtle import pd
import time
#----------------------Method for printing arrays
    ------------------------------

def arrayPrint(matrix):
    s = [[str(e) for e in row] for row in matrix]
    lens = [max(map(len, col)) for col in zip(*s)]
    fmt = '\t'.join('{{:{}}}'.format(x) for x in lens)
    table = [fmt.format(*row) for row in s]
    print('
        ------------------------------------------------------------------
        ')
    print('\n'.join(table))
    print('
        ------------------------------------------------------------------
        ')
#---------------Implementation of Recursive -------------------------
def recurit(i,j):
    #------------Base Case---------
    #print("Called")
    if i == j:
        result = val[i]
        return result
    if j == i+1:
        result = max(val[i],val[j])
        return result
    #-----Recursive Algorithim----
    result = max(val[i] + min(recurit(i+2,j), recurit(i+1,j-1)), val[j] + min(recurit(i+1,j-1),recurit(i
        ,j-2)))
    print(result)
#-----------------Implementation of Dynamic Programming -----------------
def timber(n,val):
#------- Build the table ----------------------
    T = [[0 for x in range(n)] for y in range(n)]
    TB = [[0 for x in range(n)] for y in range(n)]

#------ Fill Table with base cases-------------
    for i in range(n-1,-1,-1):
        for j in range(n):
            if j >= i:
                if j == i:
                    T[i][j] = val[i]
                if j == i+1:
                    T[i][j] = max(val[i],val[j])
#---------Calculte remaining items ------------
    sel = []
    for i in range(n,-1,-1):
```

```
        for j in range(n):
            if j >= i and T[i][j] == 0:

                T[i][j] = max(val[i] + min(T[i+2][j], T[i+1][j-1]), val[j] + min(T[i+1][j-1],T[i][j-2]))

                #------ Logic For Filling Traceback Table ----------
                ii = val[i] + T[i+2][j]
                ij = val[i] + T[i+1][j-1]
                ji = val[i] + T[i+1][j-1]
                jj = val[j]+ T[i][j-2]

                if min(ii,ij)>min(jj,ji):
                    if ij<ii:
                        TB[i][j] = "lr"
                    else:
                        TB[i][j] = "ll"
                else:
                    if ji<jj:
                        TB[i][j] = "rl"
                    else:
                        TB[i][j] = "rr"
    i = 0
    j = n - 1

    while i < j:
        if TB[i][j] == "ll":
            sel.append(i+1)
            #val.pop(i+1)
            i = i+1
            sel.append(i+1)
            #val.pop(i+1)
            i = i+1
        if TB[i][j]:
            sel.append(i+1)
            #val.pop(i+1)
            i = i +1
            sel.append(j+1)
            #val.pop(len(val)-1)
            j = j - 1
        if TB[i][j] == ("jj"):
            sel.append(j+1)
            #val.pop(len(val)-1)
            j = j - 1
            sel.append(j+1)
            #val.pop(len(val)-1)
            j = j-1
        if TB[i][j] == ("ji"):
            sel.append(j+1)
            #val.pop(len(val)-1)
            j = j - 1
            sel.append(i+1)
            #val.pop(i+1)
            i = i+1
        if TB[i][j] == ("i"):
```

```
            sel.append(i+1)
            #val.pop(i+1)
            i = i+1
        if TB[i][j] == ("j"):
            sel.append(j+1)
            #val.pop(len(val)-1)
            j = j-1
        else:
            sel.append(i+1)
            i= i+1
    summation = sum(range(1,n+1))
    zipper = sum(sel)
    missing = summation - zipper
    print(T[0][n-1])
    if missing > 0:
        sel.append(missing)
    print(sel)

#------------------This is used for building the "tree"---------------
n = int(input("Enter the number of segments on the tree(n)\n"))
mor = input("Do you want to fill your tree with manual or random values (m/r):\n")
    #manual
val=[]
lookback = []
rat = 0
if mor == 'r':
    while rat<n:
        val.append(random.randint(1,100))
        rat=rat+1
elif mor == 'm':
    while rat<n:
        userInt = int(input("Enter value for the %s segment" %(rat+1)))
        val.append(userInt)
        rat=rat+1
else:
    print("Invalid input")
    exit()
print("Here is your tree:", val)
#
    -----------------------------------------------------------------------

#----------DP //comment out if unwanted -------
timber(n,val)
#---- Recursive //comment out if unwanted------
#i = 0
#j = n-1
#result = recurit(i,j)
#print("Recursive ", result)
#------------------ The End ---------------------
```