



Name: Prudhvi Reddy Araga

CSU ID: praraga

0/70

WARNING: Search Internet

Question-1:

- a) Code for inner product of vectors X and Y **WARNING**

<https://www.w3resource.com/python-exercises/numpy/basic/numpy-basic-exercise-33.php>

```
import numpy as np
x = np.array([4, 5, 6, 8, 9])
y = np.array([7, 10, 1, 2, 3])
print("Original vectors:")
print(x)
print(y)
print("Inner product of said vectors:")
print(np.dot(x, y))
```

Output:

```
Original vectors:
[4 5 6 8 9]
[ 7 10  1  2  3]
Inner product of said vectors:
127
```

The time complexity of inner product of two vectors is $\theta(n)$.

- b) **Recursive** function to calculate inner product of X and Y using divide and conquer approach.

```
int innerproduct(int vector_x[], int vector_y[])
{
    int product = 0;
    for (int i = 0; i < size; i++)
        product = product + vector_x[i] * vector_y[i];
    return product;
}

int main()
{
    int vector_a[] = { 4, 2, -1 };
    int vector_b[] = { 5, 7, 1 };
    int temp[size];
    cout << "Inner product:";
    cout << innerproduct(vector_a, vector_b) << endl;
    for (int i = 0; i < size; i++)
        cout << temp[i] << " ";
    return 0;
}
```

Not a recursive function

The recurrence relation for the above program is as follows:

$$T(n) = 2T(n/2) + 1$$

Using the masters theorem, the values are $a = 2$, $b = 2$, $c = 1$

The time complexity is $\theta(n \log n)$ [This is based on equation 4.7 from textbook].

Question – 2:

The algorithm that is used for finding the median of $2n$ elements is as follows:

The middle element of X and Y arrays:

$X[1..n]$, $Y[1..n]$

if $(X == Y)$ then this is the median

if $(X[\text{midsize}] < Y[\text{midsize}])$, then the median is median of elements from left side of X and elements from right side of Y.

explain?

if $(X[\text{midsize}] > Y[\text{midsize}])$, then the median is median of elements from left side of Y and elements from right side of X.

Algorithm findMedian(X, Y, m):

```

if m == 1,
    return (X[0] + Y[0]) / 2
midsize = [n/2]
    if X[midsize] == Y[midsize]
        return X[mid]
    if X[midsize] < Y[midsize]
        return findMedian(X[0..midsize], Y[midsize+1..n])
    return findMedian(Y[0..midsize], X[midsize+1..n])

```

Therefore, time complexity:

$$T(n) = T(n/2) + O(1)$$

As per the master theorem

$a = 1$, $b = 2$, and $f(n) = O(1)$ case II applies

Therefore, $T(n) = O(n \log n) = O(\log n)$

?

Question – 4:

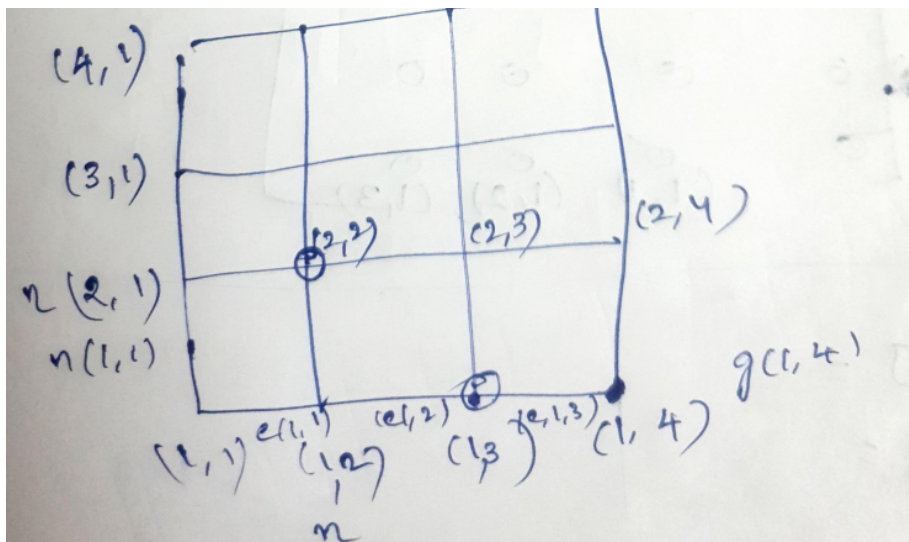
$$\text{Matrix } n = \begin{bmatrix} 7 & 4 & 8 & 17 \\ 16 & 17 & 11 & 23 \\ 15 & 12 & 18 & 11 \\ 11 & 12 & 13 & 14 \end{bmatrix}$$

$$\text{Matrix } e = \begin{bmatrix} 5 & 10 & 5 \\ 10 & 15 & 12 \\ 20 & 23 & 17 \\ 10 & 12 & 15 \end{bmatrix}$$

From $g[1][1]$, there are two paths. We can move to either $g[1][2]$ or $g[2][1]$. Similarly, there are two ways to move from any one place to the next place and so on.

So, to reach $g[m][m]$, it should reach from $g[m-1][m]$ or $g[m][m-1]$.

Cost to move all intermediate are accumulated and at each step based on minimum cost to reach from start point to current position, path is chosen.



$$g[i][j] = \sum_{i=2, j=2}^{n,n} \min(g[i][j-1] + e[i][j-1], g[i-1][j] + n[i-1][j])$$

Use cut-and-paste to justify

$$g[i][j] = g[i-1][j] + n[i-1][j] \quad j = 1, j > 1$$

$$g[i][j] = g[i][j-1] + e[i][j-1] \quad i = 1, j > 1$$

$$g[i][j] = 0 \quad i = 1, j = 1$$

b. Shortest Path (n,e):

initialize ~~$g[m][m]$~~ to 0

initialize $path[m][n]$ to (0,0)

$i = 1$

for j in range of (m)

$$g[i][j] = g[i][j-1] + e[i][j-1]$$

$$path[i][j] = (i, j-1)$$

$j = 1$

for i in range(0, m)

$$g[i][j] = g[i-1][j] + n[i-1][j]$$

$$path[i][j] = (i-1, j)$$

for i in range (1, m) : $\rightarrow m$ times

for j in range (1, m): $\rightarrow m$ times

indent if($g[i][j-1] + e[i][j-1] > g[i-1][j] + n[i-1][j]$)

$$g[i][j] = g[i-1][j] + n[i-1][j]$$

$$path[i][j] = (i-1, j)$$

else

$$g[i][j] = g[i][j-1] + e[i][j-1]$$

$$path[i][j] = (i, j-1)$$

j

$g[i][j]$ contains mincost and $path[i][j]$ contains previous node and by traversing from $path[i][j]$ to $path[0][0]$, we can get shortest path.

~~Initialize shortest $[n]$ to (0,0)~~

c. Time Complexity is $O(m^2)$ where, m is the size or length of the matrix.

There will be m^2 ways or paths in which we can travel from source to destination i.e., rows*columns.