CSE 321

HW_4 Solutions

1-

```
Algorithm BruteForceStringMatch(T[0:n-1], P[0:m-1])
    for i= 0 to n-m:
        j = 0
        while j < m and P[j] = T[i+j] :
            j++
    end for
    if j = m then return i
        return -1
    end if
```

Given text full of zero = 0000000000000....00 (n zero)

Searched pattern = 0010 (length = 4)

0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0
0	0	1	0												
	0	0	1	0											
		0	0	1	0										
			0	0	1	0									
				0	0	1	0								
					0	0	1	0							
							0	1	0						
							0	0	1	0					
											...				
												0	0	1	0

Comparasion Calculation ;

Searched text length is 4, given text size n then **n-4+1 = n-3** comparison.

0010, 3 comparasion per string comparasion (0,0,1) so total comparasion = **3(n-3) = 3n-9** comparasion.

Worst case calculation for input pattern of length 3 ;

Worst case calculation for general n and m ;

$$\sum_{i=0}^{n-m} \sum_{j=0}^{m-1} 1 = \sum_{i=0}^{n-m} m \;;;$$

$$= m(n-m+1) = m.n - m^2 + m$$

For m =3 then number of comparasion must be ;

Number of comparasion = $3.n-3^2+3$ = **$3n-6$**

2- When i applied brute force algorithm for the travelling salesman problems, the result was as follows. Algorithm found the shortest path, following all the paths. As an example, I showed the paths starting from A and D points.

```
A->B->C->D->E->A = 22
A->B->C->E->D->A = 25
A->B->D->C->E->A = 21
A->B->D->E->C->A = 27
A->B->E->C->D->A = 16
A->B->E->D->C->A = 19
A->C->B->D->E->A = 27
A->C->B->E->D->A = 22
A->C->D->B->E->A = 18
A->C->D->E->B->A = 19
A->C->E->B->D->A = 21
A->C->E->D->B->A = 27
A->D->B->C->E->A = 24
A->D->B->E->C->A = 21
A->D->C->B->E->A = 16
A->D->C->E->B->A = 16
A->D->E->B->C->A = 22
A->D->E->C->B->A = 25
A->E->B->C->D->A = 16
A->E->B->D->C->A = 18
A->E->C->B->D->A = 24
A->E->C->D->B->A = 21
A->E->D->B->C->A = 27
A->E->D->C->B->A = 22

Shortest route:16 A->B->E->C->D->A
```

```
D->A->B->C->E->D = 25
D->A->B->E->C->D = 16
D->A->C->B->E->D = 22
D->A->C->E->B->D = 21
D->A->E->B->C->D = 16
D->A->E->C->B->D = 24
D->B->A->C->E->D = 27
D->B->A->E->C->D = 21
D->B->C->A->E->D = 27
D->B->C->E->A->D = 24
D->B->E->A->C->D = 18
D->B->E->C->A->D = 21
D->C->A->B->E->D = 19
D->C->A->E->B->D = 18
D->C->B->A->E->D = 22
D->C->B->E->A->D = 16
D->C->E->A->B->D = 21
D->C->E->B->A->D = 16
D->E->A->B->C->D = 22
D->E->A->C->B->D = 27
D->E->B->A->C->D = 19
D->E->B->C->A->D = 22
D->E->C->A->B->D = 27
D->E->C->B->A->D = 25

Shortest route: 16 D->A->B->E->C->D
```

Using again brute-force algorithm then result for other start points;

Route for start point B = 16 B-> A-> D -> C-> E-> B

Route for start point C = 16 C ->B-> E-> A-> D-> C

Route for start point E = 16 E-> A-> D-> C-> B-> E

3- I designed an algorithm that finds the index of a number in a sorted array.

```
find_indexOf_number (A[0:N],low,high,x)

    if low > high :
        return -1

    mid = (low+high)/2
    if x == A[mid] :
        return mid
    else if A[mid] > x :
        return find_indexOf_number(A,low,mid-1,x)
    else
        return find_indexOf_number(A,mid+1,high,x)

end
```

Recurrence relation of this algorithm;

$T(n) = T(n/2) + O(1)$ then using master's theorem;

$A = 1$; $B = 2$; and $f(n) = O(1)$ (constant)

we have $f(n) = O(1) = O(n^{\log_b a})$

then result is = $T(n) = O(n^{\log_b a} \log_2{}^{n}) =$ **$O(\log_2{}^{n})$**

4-To solve this problem, i used decrease and conquer algorithm, i do first divide 2 all of bootles then checking total mass of each group and check bootle count.

Assume there are n bottles an done of them has a different weight

5	5	7	5	5	5	5	5

N=8

If n is even then divide into n/2 piles

5	5	7	5

n/2 =4

5	5	5	5

n/2 =4

If n is odd then divide into n/2+1 piles if n =7

5	5	7	5

n/2 =4

5	5	5

(n-1)/2 =3

we calculate weight the two piles seperately

5	5	7	5

Weight=22

5	5	5	5

Weight=20

5	5	7	5

5*4=20!= 22

5	5	5	5

5*4=20=20

IF > the results of the multiplication are different the bottle with a different weight is the first element of one of these two piles.

The weights of these two bottles are compared with another bottle and the different one is found.

ELSE -> the results of the multiplication are same then we compare the values we multiplied with the weights of the piles. The foulty product is in the pile where two values different.

IF > one element with different weight cannot be found it continues in the same way until n=2 -> until there are two bottles left. The weights of these two bottles are compared with another bottle and the different one is found.

Best case:The bootle of different weight is first element of array or middle element of these array it is found in the first comparision. This process takes places at a constant time O(1)

Worst case: if no element is found until n=2 -> **O(logn)**

5-

In order to solve this problem by divide and conquer algorithm, I first sorted the each array up to the x^{th} index.Because in the worst case, it will be sufficient to look up to the lowest x number in both arrays to find the number we are looking for. I used selection sort as the sorting algorithm because selection sort brings the smallest number to the right position each time.

```
procedure find_xth_num(A[m],B[n],x):
    if m+n < x :
        return INF
    a1,a2 =x
    if a1 > m :
        a1 = m
    if a2 > n :
        a2 = n
    selection_sort(A,a1)
    selection_sort(B,a2)
    return find_xth_num2(A[0:a1],B[0:a2],x)
end
```

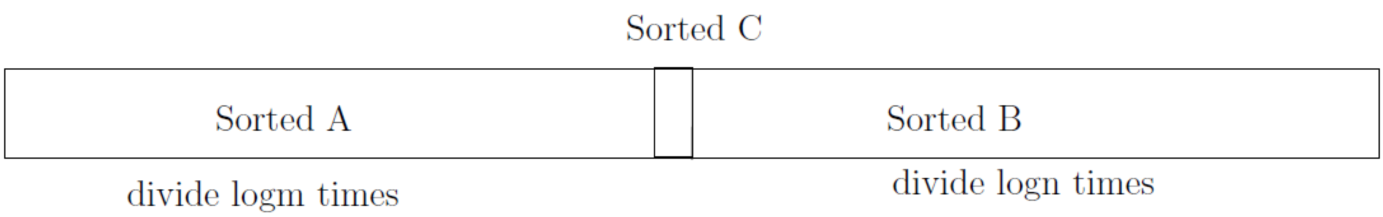
```
procedure selection_sort(A[m],x):
    for i = 0 to x:
        min_idx = i
        for j=i+1 to m:
            if A[min_idx] > A[j]:
                min_idx = j
        end for
        A[i], A[min_idx] = A[min_idx], A[i]
    end for
end
```

After sorting, we can use divide and conquer algorithm.

Let mid_a and mid_b be the middle element of array A and B respectively and let C be the sorted array of merging A and B. If $mid_b < mid_a$ and $m/2 + n/2 < k$ then k is between mid_b and the end of arrays.If $m/2 + n/2 > k$ it is from starts to mid_a the reverse is true otherwise. We can apply the previous statement by swapping a and b. Therefore we will have this algorithm.

```
procedure find_xth_num2(A[m] ,B[n] ,x):
    if A is empty:
        return B[x]
    if B is empty
        return A[x]
    else:
        if m/2+n/2 <= x:
            if A[m/2]>B[n/2]:
                find_xth_num2(A[0:m] ,B[n/2+1:n] ,x–n/2-1)
            else:
                find_xth_num2(A[m/2+1:m] ,B[0:n] ,x–m/2-1)
        else:
            if A[m/2]>B[n/2]:
                find_xth_num2(A[0:m/2] ,B[0:n] ,x)
            else:
                find_xth_num2(A[0:m] ,B[0:n/2] ,x)
end
```

The worst case happens for find_xth_num2 function, when the combination of A and B are not for example when the largest of A is smaller that B and k is one of those elements. This will make the algorithms divide both arrays equally until they are empty.For example in the figure below A will be divided log n times and B will be divided logm times to reach x.



Therefore time complexity of find_xth_num2: T(m; n) ∈ θ (logm + log n)

The Selection sort worst case complexity is O(x.n). (n is array size) then worst time complexity;

T(m;n) ∈ θ (xm+xn)+ θ (logm + log n) = θ (xm+xn+logm+logn)