

CSE221

Assignment 2

Mazidul Islam Kabbo

24301506

```

1. int func1(int n) { —  $T(n)$ 
    if ( $n \leq 1$ )
        { return 1; }
    else {
        int a = 0;
        for (int i = 0; i < 3; i++) { — 3 times
            a += func1(n/2); —  $T(n/2)$ 
        }
        int b = func2(n); —  $O(n^2)$ 
        return a + b;
    }
}

```

$$T(n) = 3T(n/2) + O(n^2)$$

```

int func2(int n)
{ int c = 0;
  for (int k = n; k >= 1; k -= 4) {
    for (int m = 0; m < n; m += 2)
        { c += k + m;
        }
  }
  return c;
}

```

comparing it to
 $T(n) = aT(n/b) + cn^k$

$$\Rightarrow \begin{aligned} a &= 3 \\ b &= 2 \\ c &= 1 \\ k &= 2 \end{aligned}$$

now,

$$b^k = 2^2 = 4, \quad a = 3$$

$$= b^k > a \Rightarrow 4 > 3$$

so,

$$T(n) \Rightarrow O(n^k) \Rightarrow O(n^2) \text{ (Ans)}$$

```

2. Function F(n) —  $T(n)$ 
    if ( $n == 1$ )
        { return True; }
    else
        { X = F(n/2) —  $T(n/2)$ 
          Y = F(n/2)
          Z = F(n/2)
          U = A(n) —  $O(1)$ 
        }
}

```

```

Function A(n)
for i = n to 1; i--
    Return True;
}

```

$$\} O(1)$$

$$T(n) = 3T(n/2) + O(1)$$

comparing it to

$$T(n) = aT(n/b) + cn^k$$

$$\begin{aligned} c &= 1 \\ k &= 0 \\ a &= 3 \\ b &= 2 \end{aligned}$$

$$b^k = 2^0 = 1 \Rightarrow b^k < a$$

$$\Rightarrow 1 < 3$$

so,

$$T(n) = O(n^{\log_b a}) \Rightarrow O(n^{1.585})$$

4. Worst case time complexity of quicksort is $O(n^2)$

- it happens when all the values are sorted in a first element pivot array.

like,

1, 2, 3, 4, 5 j upto last

↑
pivot

j

→ compares and finds no suitable value to swap with, so pivot doesn't move.

then,

2, 3, 4, 5 j upto last

↑
pivot

j

→ same again, with no suitable swaps and this pivot changes n-times with comparisons being made n times, so $O(n^2)$

5. a) We can use Kadane's algorithm

```
int bestsum (int[] A)
```

```
{ best = -∞;
```

```
sum = 0
```

```
for (int i = 0; i < A.length; i++)
```

```
{ sum = Math.max(sum + A[i], A[i]);
```

```
best = Math.max(best, sum);
```

```
}
```

```
return best; → this gives team's best consecutive streak
```

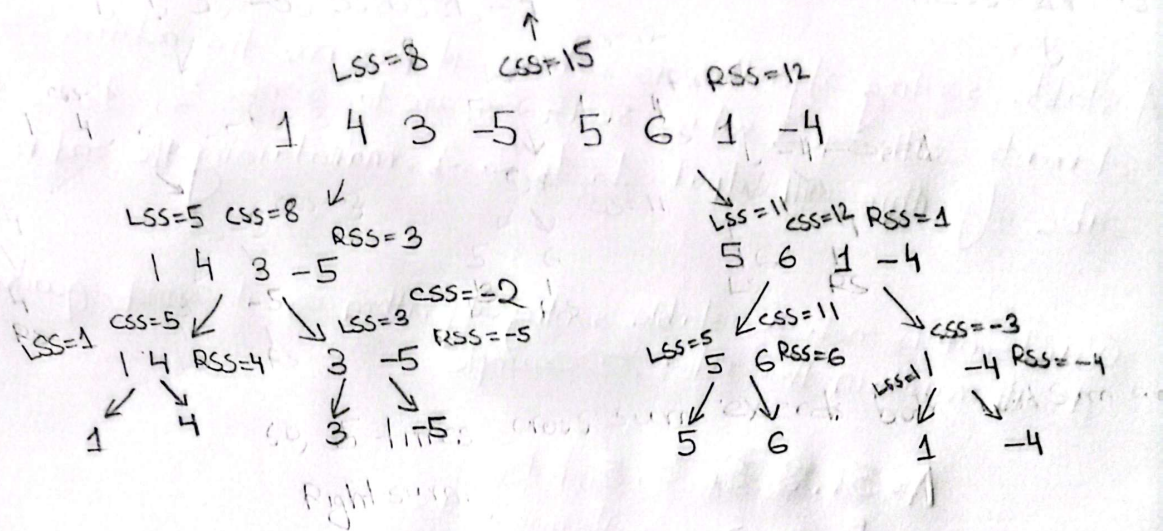
```
}
```

i.i)

1, 4, 3, -5, 5, 6, 4, -4

i	sum	best
0	1	1
1	5	5
2	8	8
3	3	8
4	8	8
5	14	14
6	15	15
7	11	15

→ best consecutive score is 15



$$n_{CSS} = 1 + 1 + 1 + 1 + 1 = 5$$

ii)

```

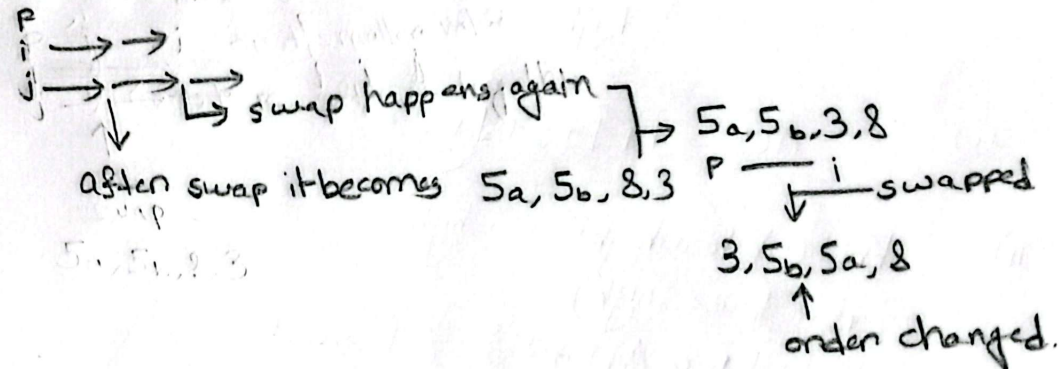
Fn1(A, low, high)
{
    if (low >= high)
    {
        return A[low];
    }
    mid = (low + high) / 2;
    LeftS = Fn1(A, low, mid);
    RightS = Fn1(A, mid + 1, high);
    crossS = Fn2(A, low, mid, high);
    return Math.max(LeftS, RightS, crossS);
}

Fn2(A, low, mid, high)
{
    LS = 0;
    sum = 0;
    for (int i = mid; i >= 0; i--)
    {
        if (A[i] < 0)
        {
            break;
        }
        sum = sum + A[i];
        LS = sum;
    }
    RS = 0;
    sum = 0;
    for (int i = mid + 1; i < A.length; i++)
    {
        if (A[i] < 0)
        {
            break;
        }
        sum = sum + A[i];
        RS = sum;
    }
    return LS + RS;
}
  
```

6. A stable sorting algorithm is where it preserves the relative order of elements with equal value, so if we are to sort, $2_a, 1_a, 2_b, 1_b$, the stable algorithm will output $1_a, 1_b, 2_a, 2_b$, maintaining the order.

Quicksort is not a stable sorting algorithm as it cannot guarantee that it will maintain the order, for example,

$A = 5_a, 8, 5_b, 3$



7. Counting Sort is suitable, as range of marks is only 0-100 and Counting Sort doesn't compare, so no comparisons and it's very fast algorithm with time complexity $O(n+k)$ where n is no. of students and k is range of input which is 0-100 or 101, so we can also say time complexity is $O(n)$ as $n \gg k$. It does use a lot of memory though but since memory is not a constraint we don't have to worry.