1)
a)
1) Bis(O) -> Upper Bound
2) Bis (Ω) -> lower Bound
3) Bis(Θ) (theta) -> Avg Bound

$$1 < \log n < n < n\log n < n^2 < 2^n < n!$$

Bis (O) UpperBound

$f(n) = g(n)$ iff $\exists$ +ve c & n_0 such that $f(n) \leq c * g(n)$ $\forall n \geq n_0$

$f(n) = 2n + 1$

$2n+1 \leq 3n+3$

$2n+1 \leq 5n^2$

$2n+1 \leq 2n \log n$

$2n+1 \leq 2^n$

$2n+1 \leq n!$

$f(n) = O(n)$
$f(n) = O(n^2)$
$f(n) = O(n \log n)$
$f(n) = O(2^n)$
$f(n) = O(n!)$

Bis Ω Lowerbound

$f(n) = \Omega(g(n))$ if $\exists$ +ve c and n_0 such that

$f(n) \geq c * g(n)$ $\forall n \geq n_0$

Es: $2n+1$

$2n+1 \geq 1 \times n$

$2n+1 \geq 1 \times \log n$

$2n+1 \geq 1$

$f(n) = \Omega(n)$
$f(n) = \Omega(\log n)$
$f(n) = \Omega(1)$

Bis (Θ) (Avg Bound)

$f(n) = \Theta(g(n))$ iff $\exists$ + ve constant c_1, c_2 and n_0 such that $c_1 * g(n) \leq f(n) \leq c_2 * g(n)$

$$f(n) = 2n+1$$

$$\underbrace{1}_{c_1} \times \underbrace{n}_{g(n)} \leq \underbrace{2n+1}_{f(n)} \leq \underbrace{5}_{c_1} \times \underbrace{n}_{g(n)}$$

$\boxed{f(n) = \Theta(n)}$

b) According to definition of Bis O, $T(f(n) + g(n)) = O(\max\{f(n), g(n)\})$ if and only if $\exists$ +ve c and n_0 such that $n \geq n_0$

$T(f(n) + g(n)) \leq c \cdot O(\max\{f(n), g(n)\})$

$f(n) \leq \max\{f(n), g(n)\}, \forall n \geq 1$

$g(n) \leq \max\{f(n), g(n)\}, \forall n \geq 1$

Thus, let $c = \cancel{2}$, $n_0 = 1$ we get that for all $n \geq 1$,

$T(f(n) + g(n)) \leq c \cdot O(\max\{f(n), g(n)\})$....

we have two loop invariants

2) ~~we use loopinvariants~~

Outer loop: Elements 1 through $i-1$ are sorted in increasing order.

Inner loop: At begining of j^{th} iteration, minIndex is the index of the smallest element in the range $[i, j-1]$.

Outerloop:

Initialization: At the beginning in the 1^{st} iteration, there not a lot of elements in the range $[1, 0]$, so they are sorted in increasing order. The second clause is also true.

Maintenance:

Suppose the invariant holds before iteration i. We show that it holds before iteration i+1 ~~We show that it holds~~. If the inner loop invariants holds, then at the end of the inner loop, minIndex is the index of the smallest element in the array range [i, A.length]. Then the swap ensures that the smallest element in the array in the range [i, A.length] gets placed in A[i]. By previous invocation of the loop invariant, we know that $A[1] \le A[2] \le \ldots \le A[i-1]$ and A[i] is atleast as large as A[i-1]. ~~Futhermore~~

Termination

At the begining of the (A.length +1)st iteration, elements 1 through A.length are sorted in increasing order, array is sorted.

Inner loop

Initialization: At the beginning of the iteration $i+1$, minIndex is i which is the index of smallest element in the range $[i, i]$

Maintenance:

Suppose the invariant holds prior to iteration j. We show that it holds prior to iteration $j+1$. At the beginning of the j^{th} iteration, minIndex is the index of the smallest element in the range $[i, j-1]$. If $A[j] < A[minIndex]$, then j is the index of the smallest element in the range $[i, j]$, so setting minIndex $= j$ is correct. If $A[j]$ is not less than $A[minIndex]$ then minIndex is the index of the smallest element in the range $[i, j]$, so it is correct to leave minIndex alone.

Termination:

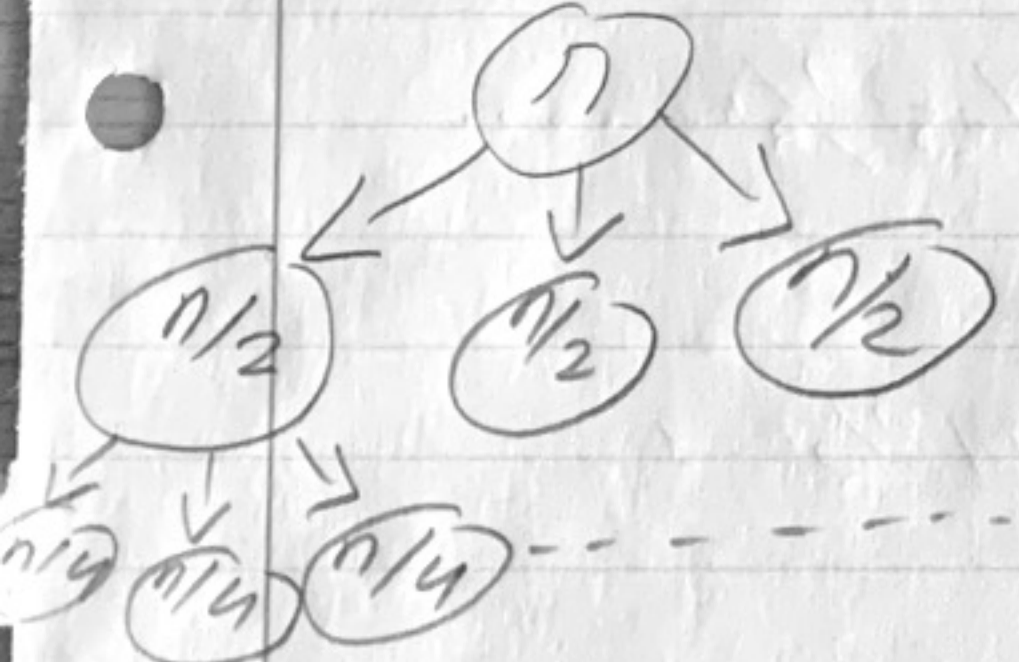
At the beginning of (A.length+1)st iteration, minIndex is the index of smallest element in the range $[i, A.length]$, which is the condition we needed for our proof of the outer loop invariant.

rec non-rec

3) $T(n) = \underbrace{3T(n/2)}_{\text{rec}} + \underbrace{2n}_{\text{non-rec}}$

n Size	Work of one call	Calls at this later	total non-recursive work at layer
n	$2n$	1	$2n$
$n/2$	$2 \cdot \frac{n}{2}$	3	$3n/2$
$n/4$	$2 \cdot n/4$	9	$9n/2$
$n/8$	$2 \cdot n/8$	$9 \cdot 3 = 27$	$27n/4$

1	$2 \cdot 1 = O(1)$	$3^{\log_2 n}$	$3^{\log_2 n}$



How manytimes did n get cut in half?

$K = \log_2 n$

$2n, 3n, 9n/2, 27n/2$

$\times 3/2 \quad \times 3/2 \quad \times 3n/2$

$S_n = \dfrac{a(r^k - 1)}{r - 1}$ $\quad r = 3/2$ $\quad a = 2n$ $\quad K = \log_2 n$

$\dfrac{2n\left((3/2)^{\log_2 n} - 1\right)}{3/2 - 1} = 4n\left((3/2)^{\log_2 n} - 1\right)$

$T(n) = 4n\left(\frac{3}{2}\right)^{\log_2 n} - 4n = 4n\dfrac{3^{\log_2 n}}{2^{\log_2 n}} - 4n$

$= \cancel{4n}\dfrac{3^{\log_2 n}}{\cancel{n}} - \cancel{4n}$

$= O\left(3^{\log_2 n}\right)$

$= O\left(n^{1.6}\right)$

4) $4T(n/2) + n^2 \log n$

$$T(n) = 4T(n/2) + n^2 \log n$$
$$= 4[4T(n/2^2) + (n/2)^2 \log n/2] + n^2 \log n$$
$$= 4^2 T(n/2^2) + 4(n/2)^2 \log n/2 + n^2 \log n$$
$$= 4^2[4T(n/2^3) + (n/2^2)^2 \log n/2^2] 4(n/2)^2 \log n/2 + n^2 \log n$$
$$= 4^3 T(n/2^3) + 4^2(n/2^2)^2 \log n/2^2 + 4(n/2)^2 \log n/2 + n^2 \log n$$
$$\vdots$$
$$= 4^k T(n/2^k) + 4^{k-1}(n/2^{k-1})^2 \log n/2^{k-1} \ldots 4(n/2)^2 \log n/2 + n^2 \log n$$

This will continue until $n/2^k = 1$,
$k = \log 2 n$

$$4^2(n/2^2)^2 \log(n/2)^2 \; 4(n/2)^2 \log(n/2) + n^2 \log n$$
$$n^2 \log (n/2)^2$$
$$n^2 (\log (n/2)^2 + \log (n/2) + \log n$$

$$T(n) = n^2 \log ((n/2)^2 \times (n/2) \times n)$$

4) b- Base Case:

$n=1$

$T(1) = C + 1^2/\log^2 1 = C$

$= O(1^2/\log 1)\ C$

$0 \leq C$

$T(k)$ is true

Then $T(k) \leq k^2 C, k^2/\log^2 K = O(k^2/\log^2 k)$

$T(k+1) = O((k+1)^2/\log^2(k+1))$

$(k+1)^2 C + (k+1)^2/\log^2(k+1) = O((k+1)^2/\log^2(k+1))$

Therefore, $T(n) \leq n^2 C + n^2/\log^2 n$;

Proof by induction: $T(n) = O(n^2/\log^2 n)$

5) $T(n) = a \cdot T(n/b) + O(n^d)$

a) $T(n) = 3T(n/4) + \sqrt{n}$

$a = 3$

$b = 4$

$d = 1/2$

$a > b^d$

$3 > 4^{1/2}$

$3 > 2$

$O(n^{\log_4 3})$

b) $T(n) = 7T(n/3) + O(n^3)$

$a = 7$

$b = 3$ $\quad a < b^d$

$d = 3$ $\quad 7 < 9$

$O(n^3)$

c) $T(n) = 2T(n/3) + n^6$

$a = 2$

$b = 3$ $\quad 2 < 3$

$d = 6$ $\quad O(n^6)$

6) Proof Master Theorm

$$T(n) = aT(n/b) + n^d \quad - 1$$

$$T(n/b) = aT\left(\frac{n/b}{b}\right) + (n/b)^d \quad - 2$$

$$T(n/b^2) = aT\left(\frac{n/b^2}{b}\right) + (n/b^2)^d$$

$$T(n/b^2) = aT(n/b^3) + (n/b^2)^d \quad - 3$$

$$T(n/b) = a\left(a \times T\left(\frac{n}{b^3}\right) + \left(\frac{n}{b^2}\right)^d\right) + (n/b)^d$$

$$T(n/b) = a^2 T(n/b^3) + a(n/b^2)^d + (n/b)^d \quad - 4$$

4 in 1

$$T(n) = a\left(a^2 T(n/b^3) + a(n/b^2)^d + (n/b)^d\right) + n^d$$

$$T(n) = a^3 T(n/b^3) + a^2(n/b^2)^d + a(n/b)^d + n^d$$

$$= a^3 T(n/b^3) + n^d\left((a/b^d)^2 + a/b^d + 1\right)$$

$$\left(\frac{a}{b^d}\right)^0$$

T(n)

3rd level

$a = b^d$ $\quad T(n) = a^{\log_b n} + n^d(k) \Rightarrow O(n^d \log n)$

$n^{\log_b a}$ $\quad n^{\log_b a} + n^d\left(\frac{1-r^k}{1-r}\right)$

$a \neq b^d$ $\to \log_b a > d \to a > b^d$

$\to \log_b a < d$

$\log_b a = d$

$c^{\log_b a} = a^{\log_b c}$ $\quad T(n) = a^k T(n/b^k) + O\left(n^d\left(1 + \left(\frac{a}{b}\right) + \dots + \left(\frac{a}{b}\right)^{k-1}\right)\right)$

$a^k + n^d\left(\frac{1-r^n}{1-r}\right)$ if $a = b^d \Rightarrow d < \log_a b$

$d > \log_a b$

$= a^k + n^d \times 1$ if $a = b^d$

$r = a/b^d$

$\frac{n}{b^2} = 1$

$n = b^2$

$k = \log_b n$

$a < b^d \Rightarrow d > \log_b a$

$T(n) = n^{\log_b a} + n^d\left(\frac{1-r^k}{1-r}\right)$

$= n^{\log_b a} + n^d\left(\frac{1-(a/b^d)^k}{1-a/b^d}\right)$ $\quad a < b^d$

$\frac{a}{b^d} < 1$

$= n^{\log_b a} + n^d(5)$

$\left(\frac{1-c^k}{1-c}\right)$ $\quad P > 1$

$= O(n^d)$

a) $a > b^d$

$\left(\frac{a}{b^d}\right)^k > 1$

$T(n) = n^{\log_b a} + n^d\left(\frac{1-a/b^d}{1-a/b^d}\right)$

7) a- Intialize Count to 0

For each user i from 1 to n $O(n^2)$

For each user j, from i+1 to n:

if (a[i] <= b[j] and a[j] <= a[i])

increment count

~~R~~return count.

Runs a double for loop and check if time intervals overlaps. Each pair of user (i, j) if their intervals overlaps start time of one user is less than or equal to the end time of another user or vice versa, then increment the count.

b-

```
Qheap = []                    O(nlogn)
Count = 0, time = 0, res = 0
while Count != len(nums):
    time += 1
    # popping
    if heap[0] == time:
        heap.pop(Qhead)
    # pushing
    if nums[cnt][0] == time
        res += len(heapq)
        Heapq.heappush(Qheap, nums[count][0])
        Count += 1
return res.
```

Algorithm iterates through each user's entrance and exit times on a social media site. It maintains a heap to track the user's present on the site at any given time. Heap add new users who have entered and removes the ones that have left. It calculates the number of distinct pairs, by adding the current heap size to the result.

This process continue until all users have been processed.