The Master Theorem- Case-II

Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence

$$aT(n/b) + f(n),$$

2. If $f(n) = \Theta(n^{\log_b a} \log^k n)$ for $k \geq 0$, then $T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$.

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

$a = 2$

$b = 2$.

$n^{\log_b a} = n^{\log_2 2} = n$

$f(n) = n$ (Case-II)

$f(n) = n \log^0 n$

$f(n)$

$\frac{f(n)}{n^{\log_b a}} = \frac{n}{n^2}$

$\frac{n}{n^2}$

$T(n) = \Theta = n^{\log_2 2} \log n$

$= n \log n$

$f(n)$ is $\Theta(n^{\log_2 2} \log^k n)$

n is $\Theta(n^{\log_2 2} \log^k n)$

The Master Theorem- Case-II

Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence

$$aT(n/b) + f(n),$$

2. If $f(n) = \Theta(n^{\log_b a} \log^k n)$ for $k \geq 0$, then $T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$.

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

Case (II) is applicable

$a = 2$

$b = 2$

$n^{\log_b a} = n$

n is $\Theta(n^{\log_2 2} \log^0 n)$ $k = 0$

$T(n) = \Theta(n^{\log_2 2} \log n)$

$= n \log n$ — height

height

Example $T(n) = 2.T(n/2) + n$ — (I)

$T(n/2) = 2T(n/2^2) + n/2$ — (II)

put (II) in (I)

$T(n) = 2\left(2T(n/2^2) + n/2\right) + n$

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

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

guess the kth term

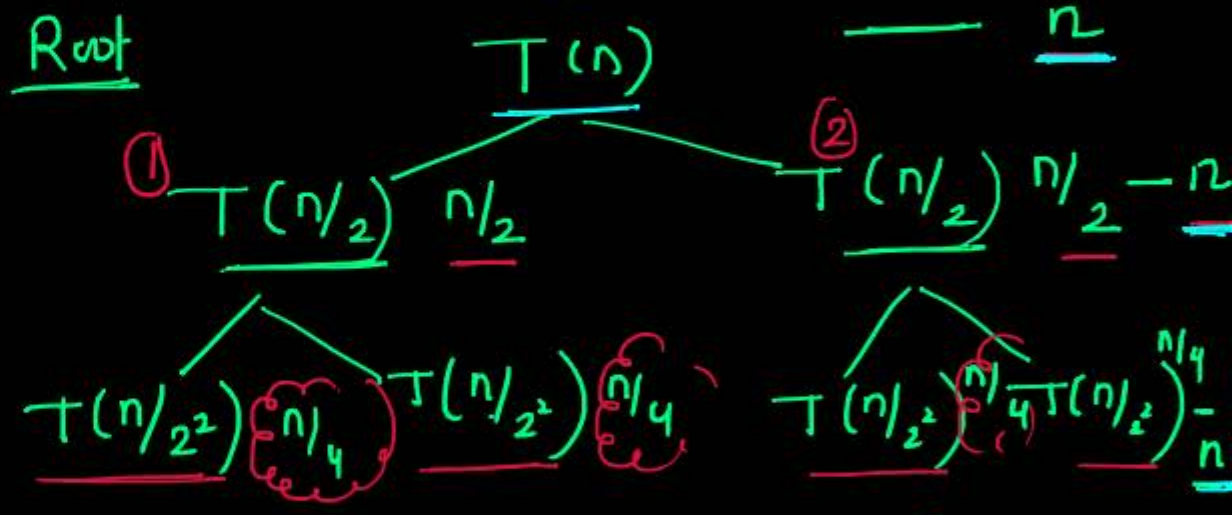
$T(n) = 2^k T(n/2^k) + kn$

Reducing to base

$n = 2^k \Rightarrow k = \log_2 n$

$T(n) = 2^k \cdot T(1) + n\log n$

$= n + n\log_2 n = \theta(n\log n)$



Height - Count the No. of Nodes

h = 0 - 1

h = 1 - 2(2¹)

at height 2 - 4(2²)

at height h - 2^h

Example $T(n) = 2.T\ (n/2) + n$

$h=0 \quad T(n)$

$h=1 \quad T(n/2)$

$h=2 \quad T(n/2^2)$

at height h - $T(n/2^h)$

$n/2^h = 1$ (Leaf Node)

$n = 2^h$ - $h = \log_2 n$

No. of leaves

at height h - 2^h

$2^{\log_2 n} = n^{\log_2 2} = n$

$n^{\log_2 2}$ ⟸ height of the tree

$n + n + n + \ldots + n$

$n(\log_2 n + 1)$

$n\log_2 n + n$ ⟸

The Master Theorem- Case-II

(Case-I)

(I) $T(n) = 25T(n/5) + n$ ($n^{\log_5 25} = n^2$, n)

(II) $T(n) = 25T(n/5) + n^2$ ($n^{\log_5 25} = n^2$, n^2) ← case (II) $k=0$

(III) $T(n) = 7T(n/6) + n$ ($n^{\log_6 7} = n^{1\cdots}$, $f(n) = n$) ← case (I)

(IV) $T(n) = 2^n T(n/4) + n$ ← Master Method Not Appliable (a, b constant)

(V) $T(n) = \dfrac{9T(n/3) + n^2}{64T(n/4) + n^3}$ (Not

(VI) $T(n) =$ ($n^{\log_3 9} = n^2$, $f(n) = n^2$) case II $k=0$

(VII) $T(n) = 2T(n/2) + n\log n$ ($n^{\log_4 64} = n^3$, $f(n) = n^3$) case III $k=0$

$T(n) = n^{\log_2 2} \log^{k+1} n$ $n^{\log_2 2} = n$, $f(n) = n\log n$

$= n \log^2 n$ $f(n) = \theta(n^{\log_2 2} \log^k n)$ $k=1$

The Master Theorem- Case-III

3. If $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$, and if $af(n/b) \leq c\, f(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.

$\sin x$, $\cos x$ (Regularity check)

$f(n)$ is a asymptotically bigger function.

$n^{\log_b a}$

Example: $T(n) = 3T(n/3) + n^2$

1-10 sec

$n^2 - n^{\log_3 3} = n$

$T(n) = \Theta(n^2)$

$T(n) = 3.T(n/3) + n^2$ — (I)

$T(n/3) = 3T(n/3^2) + (n/3)^2$ — (II)

5-6 mins

put (II) in (I)

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

$= 3^2T(n/3^2) + 3 \cdot \frac{n^2}{3^2} + n^2$

$= 3^2T(n/3^2) + \frac{n^2}{3} + n^2$

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

$T(n) = 3^3T(n/3^3) + \frac{n^2}{3^2} + \frac{n^2}{3} + n^2$

$T(n) = 3^kT(n/3^k) + \frac{n^2}{3^{k-1}} + \cdots + \frac{n^2}{3} + n^2$

Reducing to base condition

$n = 3^k$ $\quad k = \log_3 n$

$n + n^2\left[1 + 1/3 + 1/3^2 + \cdots 1/3^{k-1}\right]$

$n + n^2\left[1 - 1/3^k\right]$ / $1 - 1/3$

$n + \frac{3}{2}n^2\left(1 - 1/n\right)$ $\quad \Theta(n^2)$

$n + \frac{3}{2}n^2 - \frac{3}{2}n = \frac{3}{2}n^2 - \frac{1}{2}n$

The Master Theorem- Case-III

Case III covers the case when the parent nodes does more work than the children node.

The Master Theorem- Extended Form

2a. If $f(n) = \Theta(n^{\log_b a} \log^k n)$ for $k > -1$, then $T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$.

Example: $T(n) = 2T(n/2) + n \log n$

$k > -1$ $k = 0, 1$.

$n \log n$ $k = 1$

The Master Theorem- Extended Form

Example: $T(n) = 2T(n/2) + n\log n$

The Master Theorem- Extended Form

(±) $k > -1$ Same as 2

2b. If $f(n) = \Theta(n^{\log_b a} \log^k n)$ for $k = -1$, then $T(n) = \Theta(n^{\log_b a} \log\log n)$.

2b $k = -1$

Memonza

$$T(n) = 2T\left(\frac{n}{2}\right) + \frac{n}{\log_2 n}$$

$$f(n) = \Theta(n^{\log_2 2} \log^{-1} n)$$

$$T(n) = \Theta(n^{\log_2 2} \log\log n)$$

$1/\log n$

$= n \log\log n$ 2a $k > -1$

$= 2$

$\Theta = (n \log\log n)$ ⇐

The Master Theorem- Extended Form

2c. If $f(n) = \Theta(n^{\log_b a} \log^k n)$ for $k < -1$, then $T(n) = \Theta(n^{\log_b a})$.

No question from Extended form

2a — $k > -1$

$= 2$

$T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$ $\frac{n}{\log n}$

2b — $k = -1$

$T(n) = (n^{\log_b a} \log \log n)$ $\frac{n}{\log n}$

2c — ~~$k \leq -1$~~ $k < -1$

$T(n) = \Theta(n^{\log_b a})$ — Theoritical

2a - $k > -1$

Same as 2

2b $k = -1$

$T(n) = \Theta(n^{\log_b a} \log \log n)$

2c - $k < -1$

$T(n) = \Theta(n^{\log_b a})$

Regularity Condition

Regularity Condition

- In case 1 and case 2 the case conditions themselves make sure that work done by children is either more or equal to the parent but that is not the case with case 3.
- In *case 3 we apply a regulatory condition* to make sure that the parent does at least as much as the children. The regulatory condition for case 3 is

$$af(n/b) <= cf(n), c < 1$$

Regularity Condition

- The equation $T(n) = T(n/2) + n(sin(n- \pi/2) + 2)$ is a example where regulatory condition makes a huge difference.
- The equation isn't satisfying case 1 and case 2. In case 3 for large values of n it can never satisfy the regulatory condition. *Hence this equation is beyond the scope of master theorem*.

Using Master Method

Example : $T(n) = 9T(n/3) + n.$

$a = 9$

$b = 3$

$n^{\log_3 9} = n^2$

$f(n) = n$

Case - II

Case-1

$f(n)$ is $O(n^{\log_b a - \epsilon})$

$T(n) = \Theta(n^{\log_3 9})$

$= \underline{\Theta(n^2)}$

Using Master Method

Example: $T(n) = T(2n/3) + 1$

$T(n) = aT(n/b) + f(n)$

$T(n) = 2T(n/2) + 1$ $a \geq 1$ — true/false

$T(n/3/2)$

$T(n/1.5)$

$a \geq 1$

$b > 1$

$f(n)$ — +ve — const

+ ~~constant~~ function

$n^{\log_{3/2} 1}$

$f(n) = 1$

$n^0 = 1$

Constant

Case

$f(n) = 1$ $T(n) = n^{\log_{3/2} 1} \log n$

Constant $k = 0$ $= \Theta(\log n)$

Using Master Method

Example: $T(n) = 3T(n/4) + n^2$

Case - III

$a = 3$

$b = 4$

$n^{\log_4 3}$ 1

$f(n) = n^2$

Using Master Method

Example : $T(n) = 2T(n/2) + n\log n$,

$a = 2$

$b = 2$

$n^{\log_2 2} = n$

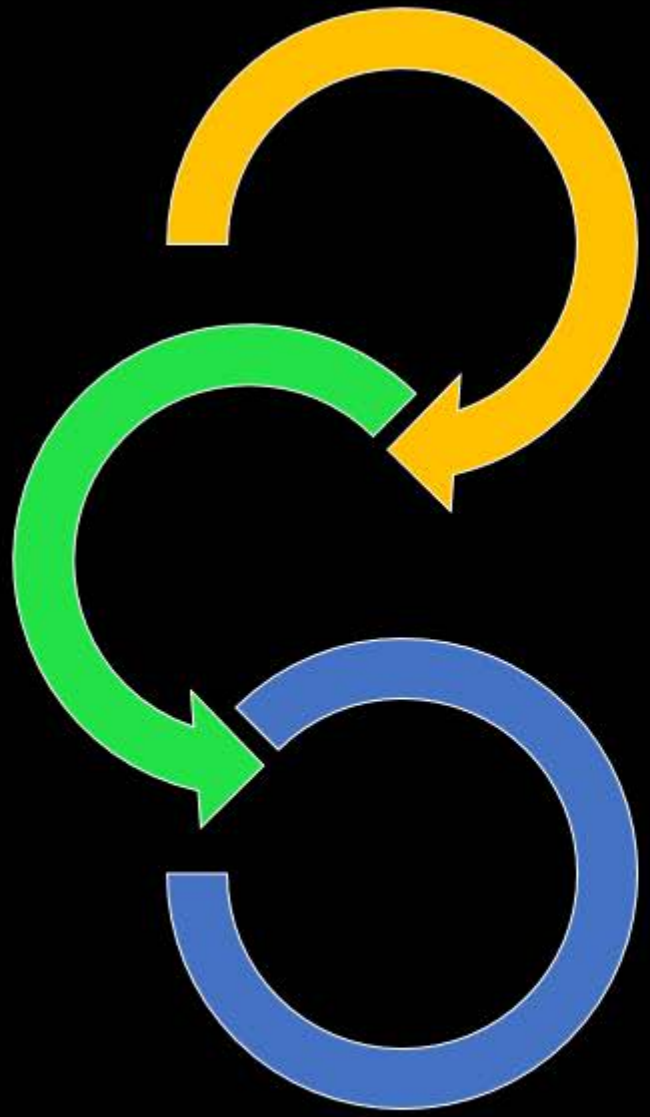
$f(n) = n \log n$

Case

$f(n)$ is $\Theta(n^{\log_b a} \log^k n)$

$T(n) = n^{\log_b a} \log^{k+1} n$

$= n \log^2 n$



Problem Solving

GATE 1987 | 2 Marks Question

Find the solution of the following recurrence relation

$$T(n)=T\left(\frac{n}{2}\right)+n$$

$$T(1)=1$$

$\Theta(n)$ Bound ?

$a=1$

$b=2$

$n^{\log_2 1} = n^0 = 1$

GATE 1996 | 2 Marks Question

The recurrence relation

$T(1) = 2$

$T(n) = 3T\left(\frac{n}{4}\right) + n$

Has the solution $T(n)$ equal to

(A) $O(n)$

(B) $O(\log n)$

(C) $O\left(n^{3/4}\right)$

(D) None of the above

a = 3
b = 4

$n^{\log_4 3}$

$n^{<1}$

$f(n)$

n

Case -

Bound

Apply master

GATE 1997 | 2 Marks Question

Let $T(n)$ be the function defined by

$T(1) = 1$, $T(n) = 2T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + \sqrt{n}$ for $n \geq 2$. Which of the following statements is TRUE?

(A) $T(n) = O\sqrt{n}$

(B) $T(n) = O(n)$ ✓

(C) $T(n) = O(\log n)$

(D) None of the above

Bound

Master Method

$a = 2$

$b = 2$

$n^{\log_2 2} = n$

$f(n) = \sqrt{n}$

Case - I

$T(n) = 2T(n/2) + \sqrt{n}$

$O(n)$

$n\sqrt{n}$

GATE 2005-IT, Question Number 51, 2-Marks

Let T(n) be function defined by the recurrence

$T(n) = 2T\left(\frac{n}{2}\right) + \sqrt{n}$ for $n \geq 2$ and

$T(1) - 1$

which of the following statements is TRUE

(A) $T(n) = \theta(\log n)$

(B) $T(n) = \theta(\sqrt{n})$

(C) $T(n) = \theta(n)$

(D) $T(n) = \theta(n \log n)$

GATE 1994 | 2 Mark Question

The recurrence relation

$$T(1) = 2$$

$$T(n) = 3T\left(\frac{n}{4}\right) + n$$

Has the solution $T(n)$ equal to

(A) $O(n)$

(B) $O(\log n)$

(C) $O\left(n^{3/4}\right)$

(D) None of the above

GATE 1999 | 2 Marks Question

If $T_1 = O(1)$, give the correct matching for the following pairs:

(M)	$T_n = T_{n-1} + n$	(U)	$T_n = O(n)$
(N)	$T_n = T_{n/2} + n$	(V)	$T_n = O(n\log n)$
(O)	$T_n = T_{n/2} + n\log n$	(W)	$T_n = O(n^2)$
(P)	$T_n = T_{n-1} + \log n$	(X)	$T_n = O(\log^2 n)$

GATE | GATE CS 1999 | Question 46

(A) M-W N-V O-U P-X

(B) M-W N-U O-X P-V

(C) M-V N-W O-X P-U

(D) M-W N-U O-V P-X

GATE 2005-IT, Question Number 51, 2-Marks

Let T(n) be function defined by the recurrence

$T(n) = 2T\left(\frac{n}{2}\right) + \sqrt{n}$ for $n \geq 2$ and

T(1) – 1

which of the following statements is TRUE

(A) $T(n) = \theta(\log n)$

(B) $T(n) = \theta(\sqrt{n})$

(C) $T(n) = \theta(n)$

(D) $T(n) = \theta(n \log n)$

GATE 2005-IT, Question Number 51, 2-Marks

Let T(n) be function defined by the recurrence

$T(n) = 2T\left(\frac{n}{2}\right) + \sqrt{n}$ for $n \geq 2$ and

$T(1) = 1$

GATE 2009, Question Number 35, 2-Marks

The running time of an algorithm is represented by the following recurrence relation:

$$T(n) = \begin{cases} n & n \leq 3 \\ T\left(\frac{n}{3}\right) + cn & otherwise \end{cases}$$

which one of the following represents the time complexity of the algorithm?

(A) $\theta(n)$

(B) $\theta(n \log n)$ (C) $\theta(n^2)$ (D) $\theta(n^2 \log n)$

$T(n) = T(n/3) + n$

$a = 1$

$b = 3$

$n^{\log_3 1} = n^0 = 1$

$f(n) = n$

GATE 2009, Question Number 35, 2-Marks

$$T(n) = \begin{cases} n & n \leq 3 \\ T\left(\frac{n}{3}\right) + cn & otherwise \end{cases}$$

GATE 2014 Set-II, Question Number 13

Which one of the following correctly determines the solution of the recurrence relation with T(1) = 1 ?

$$T(n) = 2T\left(\frac{n}{2}\right) + \log n$$

(A) $\theta(n)$ (B) $\theta(n \log n)$ (C) $\theta(n^2)$ (D) $\theta(\log n)$

(A)

$a = 2$

$b = 2$

$n^{\log_2 2} = n$

$f(n) = \log n$

GATE 2016 Set-II, Question Number 39, 2-Marks, Question Category-NAT
Subject: *Algorithms*, Topic: *Divide & Conquer*

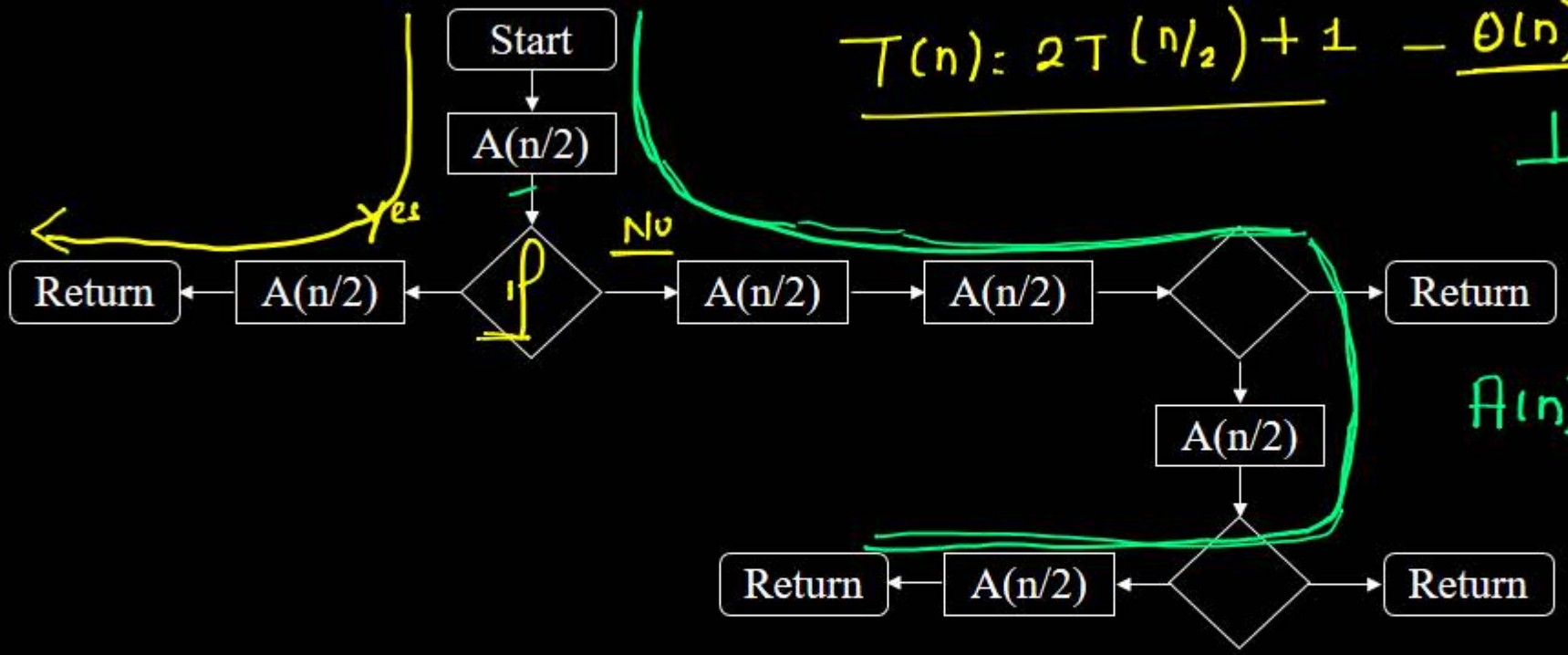
The given diagram shows the flowchart for a recursive function A(n). Assume that all statements, except for the recursive calls, have 0 (1) time complexity. If the worst case time complexity of this function is $O(n^{\alpha})$, then the least possible value (accurate up to two decimal positions) of α is _____.

O(1) – Except Recursive

Suppose worst case time complexity

$T(n) = 2T(n/2) + 1$ – $\Theta(n)$

Longest path taken



worst case

$A(n) = 5T(n/2) + 1$

worst case time complexity

$O(n^a)$ – least value of a

$\Theta(n^{2.32})$

$a = 5$

$b = 2$

$n^{\log_2 5}$

$f(n) = 1$

$\frac{\log 5}{\log 2} = \frac{.6989}{.3010} = 2.32$

GATE 2008-IT, Question Number 44, 2-Marks, Question Category-MCQ
Subject: *Algorithms*, Topic: *Recurrence Relation*

When $n=2^k$ for some $k \geq 0$, the recurrence relation

$T(n) - \sqrt{2}\, T(n/2) + \sqrt{n}, T(1) = 1$ evaluates to

(A) $\sqrt{n}\,(\log n + 1)$ (B) $\sqrt{n} \log n$ (C) $\sqrt{n} \log \sqrt{n}$ (D) $n \log \sqrt{n}$

T(n) (A)

$$T(n) = \sqrt{2}\,T(n/2) + \sqrt{n} \quad \text{—— (I)}$$

$$T(n/2) = \sqrt{2}\,T(n/2^2) + \sqrt{\frac{n}{2}} \quad \text{—— (II)}$$

put (II) in (I)

$$T(n) = \sqrt{2}\left(\sqrt{2}\,T(n/2^2) + \frac{\sqrt{n}}{\sqrt{2}}\right) + \sqrt{n}$$

$$= (\sqrt{2})^2\,T(n/2^2) + 2\sqrt{n}$$

guess the kth term

$$T(n) = (\sqrt{2})^k\,T(n/2^k) + k\sqrt{n}$$

Reducing to base condition

$n = 2^k \Rightarrow k = \log n$

$$(\sqrt{2})^{\log_2 n}\,T(1) + \sqrt{n}\log n$$

$$2^{1/2 * \log_2 n}$$

$$2^{\log_2 \sqrt{n}} = \sqrt{n}$$

$$\sqrt{n} + \sqrt{n}\log_2 n$$

$$\sqrt{n}(\log_2 + 1)$$

GATE 2021 SET-II| 2 Mark Question

For constants $a \geq 1 \text{ and } b > 1$, consider the following recurrence defined on the non-negative integers:

$$T(n) = a.T\left(\frac{n}{b}\right) + f(n)$$

Last year

Which one of the following options is correct about the recurrence T(n)?

(A) $If\ f(n) is \frac{n}{\log_2(n)}, then\ T(n) is\ \Theta(\log_2(n))$

(B) $If\ f(n) is\ \Theta\left(n^{\log_b(a)}\right), then\ T(n) is\ \Theta\left(n^{\log_b(a)}\right)$ ✗

(C) $If\ f(n) is\ O\left(n^{\log_b(a)-\epsilon}\right) for\ some \in > 0, then\ T(n) is\ \Theta\left(n^{\log_b(a)}\right)$ ✓ Case-I

(D) $If\ f(n) is\ n\log_2(n), then\ T(n) is\ \Theta(n\log_2(n))$ ✗

Inadmissible Equations

The following equations cannot be solved using master theorem:[4]

- $T(n) = 2^n T\left(\frac{n}{2}\right) + n^n$

  a is not a constant; the number of subproblems should be fixed

- $T(n) = 2T\left(\frac{n}{2}\right) + \left(\frac{n}{\log n}\right)$

  non-polynomial difference between f(n) and $n^{\log_b a}$

  (see below; extended version applies)

- $T(n) = 0.5T\left(\frac{n}{2}\right) + n$

  $a < 1$ cannot have less than one sub problem

$k > -1$ –

$k = -1$ – 2b

$k < -1$ – 2c

– 2b

– 2b $\left(T(n) = \Theta(n \log\log n)\right)$

at least 1 sub problem

Inadmissible Equations

- $T(n) = 64T\left(\frac{n}{2}\right) - n^2 \log n$

  F(n), which is the combination time, is not positive

- $T(n) = T\left(\frac{n}{2}\right) + n(2 - \text{coz } n)$

  case 3 but regularity violation.

+

Soluhon

Add some time

time Complexity

$T(n) = 2T(n/2) \pm$

Adclihon

Constant

Practice Question

Problem-1: $T(n)= 3T(n/2)+n^2$ — $n^{\log_2 3}$ — $\theta(n^2)$ case-III

Problem-2: $T(n)= 4T(n/2)+n^2$ — $T(n) = \theta(n^2 \log n)$ — case-II

Problem-3: $T(n)= T(n/2)+2^n$ — $T(n) = \theta(2^n)$ — Case-III

Problem-4: $T(n)= 2^n T(n/2)+n^2$ — Not applicable

Problem-5: $T(n)= 16T(n/4)+n$ — $T(n) = \theta(n^2)$ — case(I)

Problem-6: $T(n)= 2T(n/2)+n\log n$ — $T(n) = n \log^2 n$

Problem-7: $T(n)= 2T(n/2)+n \log n$

Problem-8: $T(n)= 2T(n/4)+n^{0.51}$ $= n^{\log_4 2} = n^{0.5}$ $f(n)\ n^{0.51}$

$=$ case III $T(n) = \theta(\phi n^{0.51})$

Practice Question

Problem-9: $T(n)= 0.5T(n/2)+1/n$

Problem-10: $T(n)= 6T(n/3)+n^2 \log n$

Problem-11: $T(n)= 64T(n/8)-n^2 \log n$

Problem-12: $T(n)= 7T(n/3)+n^2$

Problem-13: $T(n)= 4T(n/2)+\log n$

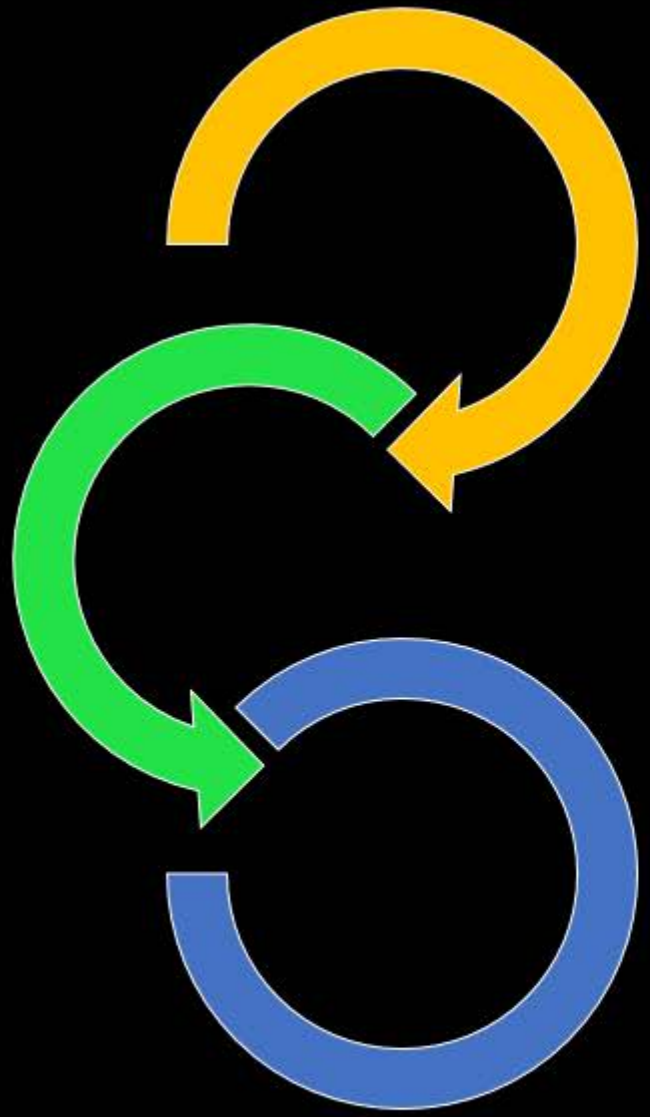
Problem-14: $T(n)= 16T(n/4)+n!$

Problem-15: $dT(n)= \sqrt{2}T(n/2)+\log n$

Problem-16: $T(n)= 3T(n/2)+n$

Lecture -5

Min Max Problem



Finding Maximum and Minimum

First problem of divide & Conquer

Unorder array

Ordered Array - where is minimum
Maximum

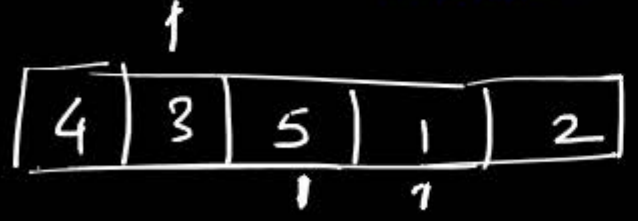
~~asced~~ ascending order - 1. min
Last max

Min and Max Problem

Given an unordered array it requires to find minimum & maximum element in array.

· Naive Method :. Sorting — Merge Sort ($n \log n$)

Scan through the List and find minimum & Maximum.



max = min = 4

max = 4 min = 3

max = 5 min = 3

min = 1

Algorithm maxmin (a, n) {

max = min = a[1];

for i = 2 to n {

if a[i] > max,

max = a[i]

else

if (a[i] < min). ; min = a[i] }

Min and Max Problem

The problem is to find the maximum and minimum items in a set of n elements. Following Algorithm is a straightforward algorithm to accomplish this.

Min and Max Problem

```
Algorithm StraightMaxMin(a,n,max,min){
    max := min := a[1];
    for i := 2 to n do{
    if (a[i] > max) then max :=a[i];
    else if (a[i] < min) then min := a[i];
    }
}
```

n-1 n-2+ = n-1

(n-1) —— 2 comparison

= 2(n-1) comparison

Worst Case

Number of comparison

Always min is updated

5, 4, 3, 2, 1

min = max = 5

① i = 2 max = 5, min = 4 } 2

② i = 3 max = 5, min = 3 } 2

③ i = 4 max = 5, min = 2 } 2

④ i = 5 max = 5, min = 1 } 2

Number of Comparison

Best case

Best case No. of comparison

1, 2, 3, 4, 5 ——

Best Case No. of Comparison is (n-1)

max = min = 1

i = 2 max 2
min = 1

O(n)

i = 3 max = 3
min = 1

i = 4 max = 4
min - 1

i = 5 max = 5
min = 1

Number of Comparison

- **Best case:** Now the best case occurs when the elements are in increasing order. The number of element comparisons is $n - 1$.

Number of Comparison

- **Worst case:**

Number of Comparison

- **Worst case:** The worst case occurs when the elements are in decreasing order. In this case the number of element comparisons is $2(n - 1)$.

Tournament Method

$n-1$ ——— $2n-2$

Tournament Method

- *An efficient algorithm*: We scan the list once, comparing *pairs of elements* (leaving out an if n is odd): we add the greater element to a MaxList and the smaller element to a MinList. Next, we do one scan of MaxList to get max and one scan of MinList to get min. Finally, if n is odd we compare an with max; if an < max we compare it with min.

Category of question
Eg sample problem

Tournament Method

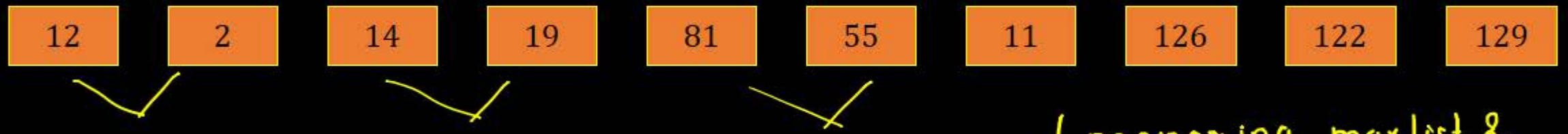
Max List

12 19 81 . 126 , 129 —

n is Even pair

Maximum will be present in maxList

Scan max List for finding maximum — $\left(\frac{n}{2}-1\right)$



Min List 2 14 55 , 11 . 122 ←

Minimum will be present in min List

Scan the minList to find minimum — $(n/2-1)$

preparing maxList & minList No. of comparison required. $n/2$

total

$\frac{n}{2}+\frac{n}{2}-1+\frac{n}{2}-1$

$= \frac{3n}{2}-2$

Number of Comparison

- *n even*:

Number of Comparison

straight minmax

n-1 — 2(n-1)

- n *even*: $n/2 + (n/2 - 1) + (n/2 - 1) = 3n/2 - 2$ *comparisons* Average.

↑ Creation of max & min List

↑ findmax from max List

↑ find minimum from List

Number of Comparison

- *n odd:* work on $n-1$ element Leaving a Single element

$$\frac{n-1}{2} + \frac{n-1}{2} - 1 + \frac{n-1}{2} - 1 + 2$$

Creating max and min list

Scan max list

So Scan the min list

Comparing max and min with 1 element left out

$$\frac{3(n-1)}{2} = \frac{3n+1}{2} - 2$$

Number of Comparison

- $n\ odd$: $\frac{n-1}{2} + \left(\frac{n-1}{2} - 1\right) + \left(\frac{n-1}{2} - 1\right) + 2 = \frac{3n+1}{2} - 2\ comparisons$ (n is odd)

$\frac{3n}{2} - 2$ (n is Even)

$\left\lceil \frac{3n}{2} \right\rceil - 2$ Comparison

optimal No. of comparison Required - $\left\lceil \frac{3n}{2} \right\rceil - 2$

Number of Comparison

- Minimum number of comparison required by optimal algorithm is $= \left\lceil \frac{3n}{2} \right\rceil - 2$

Number of Comparison

- Minimum number of comparison required by optimal algorithm is $= \left\lceil \frac{3n}{2} \right\rceil - 2$

Number of Comparison

- n *even*: $n/2 + (n/2 - 1) + (n/2 - 1) = 3n\,2 - 2$ *comparisons*

- n *odd*: $\frac{n-1}{2} + \left(\frac{n-1}{2} - 1\right) + \left(\frac{n-1}{2} - 1\right) + 2 = \frac{3n+1}{2} - 2$ *comparisons*

- Thus in any case, the number of comparisons done $= \left\lceil \frac{3n}{2} \right\rceil - 2$

GATE 2014 Set-I, Question Number 39

The minimum number of comparisons required to find the minimum and the maximum of 100 numbers is (worst case) 148.

$$\frac{3 \times 100}{2} - 2$$

$$150 - 2$$

GATE 2015 Set-II, Question Number 6

An unordered list contain n distinct elements. The number of comparisons to find an element in this list that is neither maximum nor minimum is

(A) Θ(n log n)

(B) Θ(n)

(C) Θ(log n)

(D) Θ(1) ←

Design

Neither minimum Nor maximum Scan

Randomly pick 3 elements a, b, c

among this one greater, one Least element and Last will Neighter greater Nor Lesser.

GATE 2007, Question Number 50, 2-Marks

$2(n-2)$

An array of n numbers is given, where n is an even number. The maximum as well as the minimum of these n numbers needs to be determined. Which of the following is TRUE about the number of comparisons needed?

$\frac{3n}{2} - 2$

$1.5n - c$

(A) At least 2n–c comparisons, for some constant c, are needed.

(B) At most 1.5n –2 comparisons are needed.

(C) At least $n\log_2 n$ comparisons are needed.

(D) None of the above.

GATE 2021 Set-I| 1 Mark Question

Let P be an array containing n integers, Let t be the lowest upper bound on the number of comparisons of the array elements, required to find the minimum and maximum values in an arbitrary array of n elements. Which one of the following choices is correct?

(A) $t > 2n-2$

(B) $t > \lceil \log_2(n) \rceil$ and $t \le n$

(C) $t > n$ and $t \le 3\left\lceil \frac{n}{2} \right\rceil$

(D) $t > 3\left\lceil \frac{n}{2} \right\rceil$ and $t \le 2n-2$

Divide & Conquer Approach

Algorithm maxmin (a, n, max, min) {

Lower index a - i

High Index a - j

Smaller problem – i = j 1 element

~~return a[i]~~ max = min = a[i]

Small proble (i = j-1) – 2 elements

if (a[i] > a[j]) {

max = a[i] , min = a[j] }

else

max = a[j] ; min = a[i]

} T(2) = 1

Base Case

power (x, n)

if (n == 1)

return x

else

return power (x, n/2) * pwr(n/2)

}

Divide & Conquer Approach

```
Algorithm MaxMin(A,i,j,max, min)
```

// a[l : n] is a global array. Parameters i and j are integers,

// $1 \leq i \leq j \leq n$- The effect is to set max and min to the

// largest and smallest values in a[i : j], respectively.

```
if (i = j)
   then max := min := a[i]; // Small(P)
```

Algorithm

Divide & Conquer Approach

```
else if (i = j − 1) then { // Another case of Small(P)
        if (a[i] < a[j]) then{
                max :=a[j];
                min := a[i];
        }
        else{
                max := a[i];
                min := a[j];
        }
}
```

T(2) = 1

divide part - mid = $\frac{i+j}{2}$

minmax (a, i, mid, max, min);

minmax (a, mid+1, j, max, min)

Combining { if (max1 > max), max = max1

if (min1 < min), min = min1

Divide & Conquer Approach

```
    else {
            // If P is not small, divide P into subproblems.
            // Find where to split the set.
            mid := [(i + j)/2];
            //Solve the subproblems.
            MaxMin(A,i, mid, max, min);
            MaxMin (A, mid + 1, j, max1, min1);
            // Combine the solutions.
            if (max < max1) then max := max1;
            if (min > min1) then min := min1;
        }
    }
```

does Not Required space other than array

Divide & Conquer approach

1. Straight min max
2. Tournament Method
3. Divide & Conquer approach.

Analysis of Max-Min(D & C)

No. of comparison

- When n is a power of two, $n = 2^k$ for some positive integer k then
- $T(n) = 2T(n/2) + 2$ and $T(2) = 1$ — (I)

Substitution Method

$T(n/2) = 2T(n/2^2) + 2$ — (II)

put (II) in (I)

$T(n) = 2\left(2T(n/2^2) + 2\right) + 2$

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

No. of comparison

Guess the k^{th} term

$T(n) = 2^k T(n/2^k) + 2^k + \cdots + 2^2 + 2$

Reducing to base condition

$\Rightarrow n/2^k = 2 \Rightarrow n = 2^{k+1}$

$2^k = n/2$

$= 2^k T(1) + \frac{2(2^k - 1)}{2-1}$

$= \frac{n}{2} + 2^{k+1} - 2 = \frac{n}{2} + n - 2$

$= \frac{3n}{2} - 2$

$\frac{3n}{2} - 2$

Tournament - $3n/2 - 2$

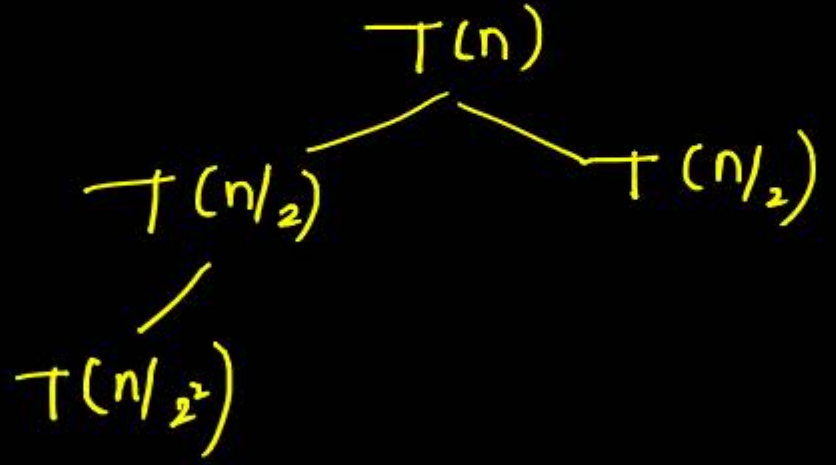
D&C - $3n/2 - 2$

Analysis of Max-Min(D & C)

- When n is a power of two, $n = 2^k$ for some positive integer k then $\Rightarrow n + \log_2 n + 1 = \theta(n)$

Space Complexity of D&C max min - Input Size - n

Stack space Required - $\log_2 n + 1$



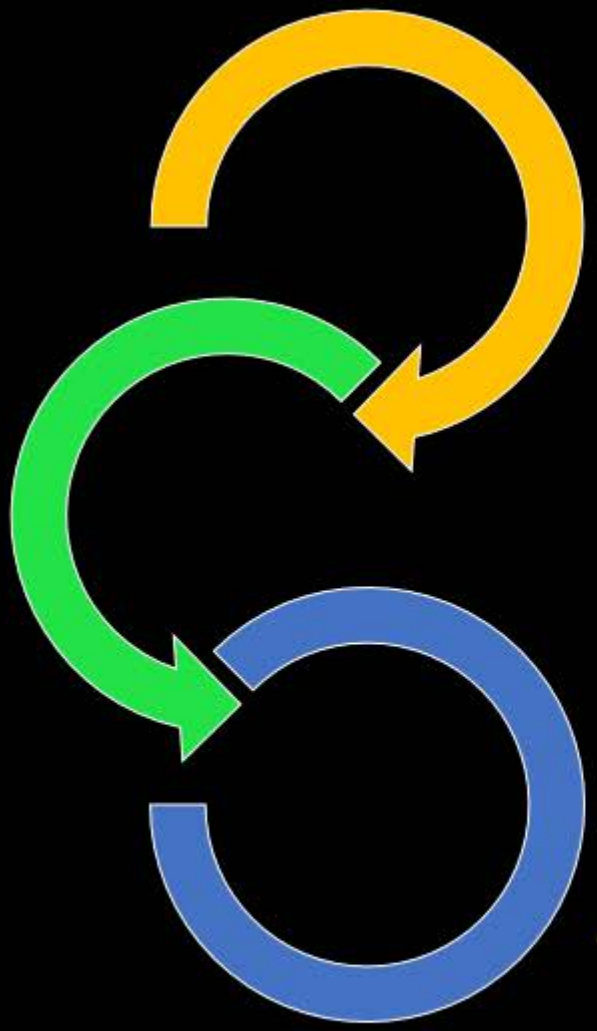
$T(n/2^k)$

$n = 2^k$

$k = \log_2 n$ Height of Recursion

Analysis of Max-Min(D & C)

- When n is a power of two, $n = 2^k$ for some positive integer k then



Quick Sort

Array is Data structure

1. Searching - Linear Search (unordered array)
   Binary Search (Ordered Array)

2. max min problem
   - straightminmax
   - Tournament Method
   - D&C approach

3. Sorting - Arranging elements in order
   1. Increasing order. 1, 2, 3, 4, 5.
   2. Non decreasing order 1, 2, 2, 3, 4, 5.
   3. Decreasing order - 5, 4, 3, 2, 1
   4. Non-Increasing Order. 5, 4, 3, 3, 2, 1

unordered array → 2 4 3 1 5 → Sorting Algm → 1, 2, 3, 4, 5

Description of Quick Sort

- Quicksort, is based on the divide-and-conquer paradigm.

A[p . . . . q]

p- Lower Bound

q- upper Bound

Basic Algorithm

1. Insertion Sort
2. Bubble Sort
3. Selection Sort

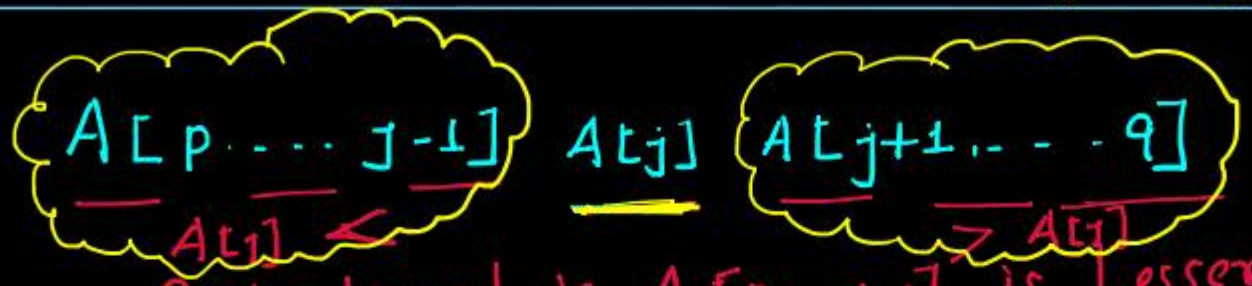
Basic Sorting

Division :

- Pivot element — Based on this pivot element the array will be divided (Partition) in 2 Sub problem or 2 Subarray.
- After ~~portion~~ partition the pivot element will be in correct position. (Suppose its index j)
- A[p ---- j-1] A[j] A[j+1 ---- q]

Three-step divide-and-conquer process

- **Divide:**



. Each element in A[p..j-1] is Lesser than A[j] (distinct)

. Each element in A[j+1 . . q] is greater than A[j]

. Each subarray will Sorted again by using quick sort

Quick(sort (A, p, j-1)

Quick sort (A, j+1, q)

. Dont Required to combining Solution because individual array is already sorted.

Three-step divide-and-conquer process

- **Divide:** Partition (rearrange) the array $A[p \quad q]$ into two (possibly empty) subarrays $A[p \quad j - 1]$ and $A[j + 1 \quad q]$ such that each element of $A[p \quad j - 1]$ is less than or equal to $A[j]$, which is, in turn, less than or equal to each element of $A[j + 1 \quad q]$. Compute the index j as part of this partitioning procedure.

Three-step divide-and-conquer process

- **Conquer:**
- **Combine:**

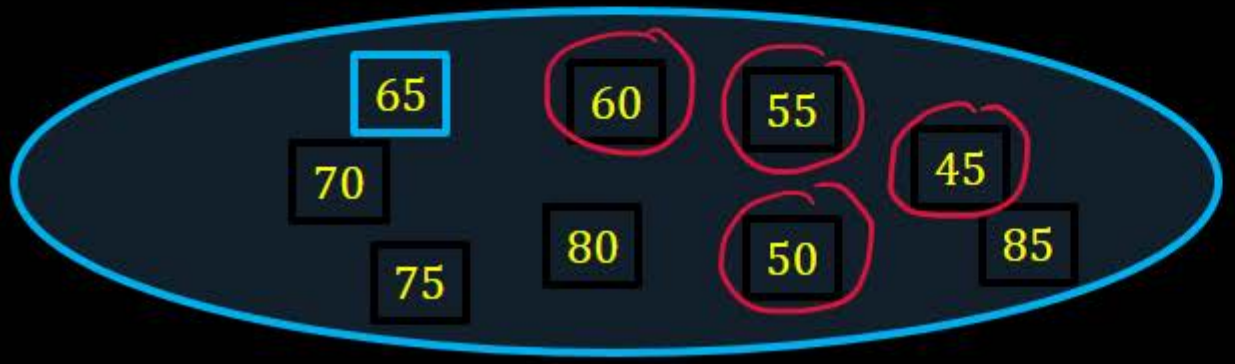
Three-step divide-and-conquer process

- **Conquer:** Sort the two subarrays $A[p \; j-1]$ and $A[j+1 \; q]$ by recursive calls to quicksort.

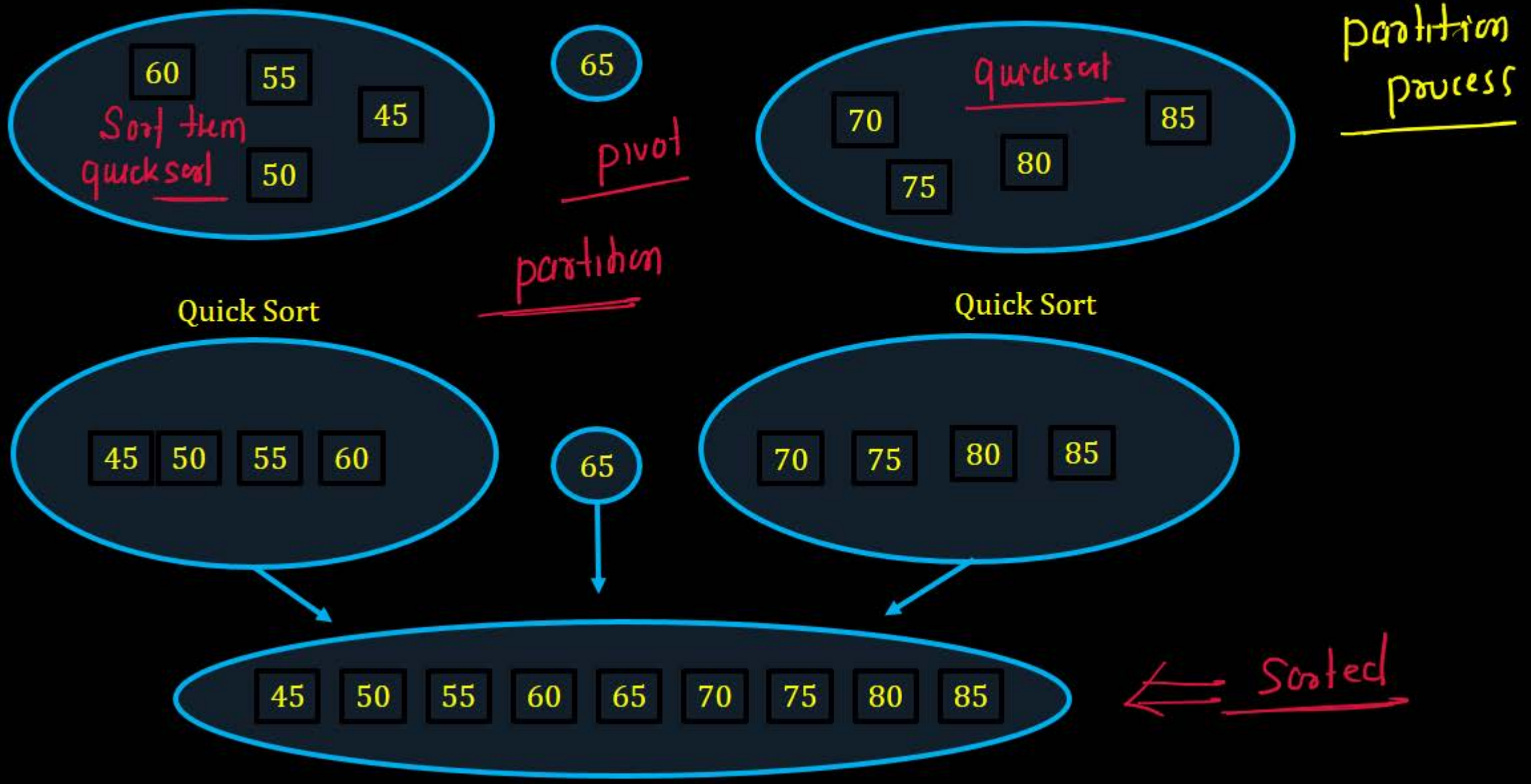
- **Combine:** Since the subarrays are sorted in place, no work is needed to combine them: the entire array $A[p \; q]$ is now sorted.

Quick Sort: Example

Select Pivot



Quick Sort: Example



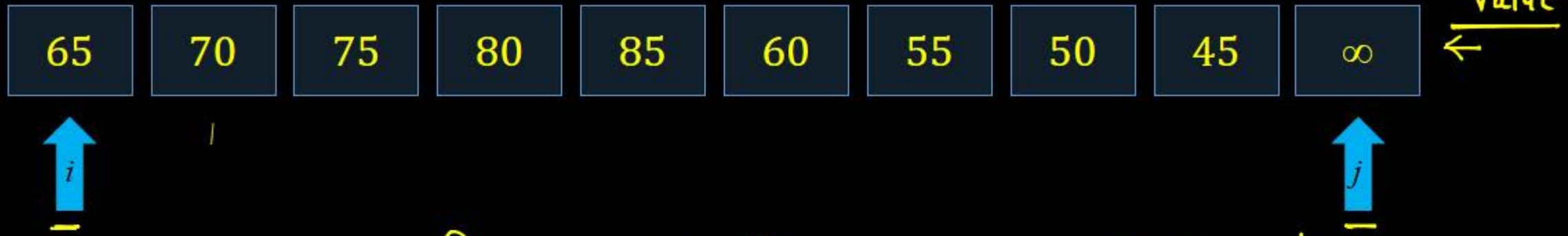
Issues To Consider

- How to pick the pivot?
  - Many methods
- How to partition?
  - Several methods exist.
  - The one we consider is known to give good results and to be easy and efficient.
  - We discuss the partition strategy first.

pivot : pivot element is the element based on which the array will be divided by partition algorithm

Hoare Method / Nico Lumuto

Hoare Partitioning



Consider the first element of array as pivot element

65 is the pivot element

two pointer

1 repeat

$i = i + 1$

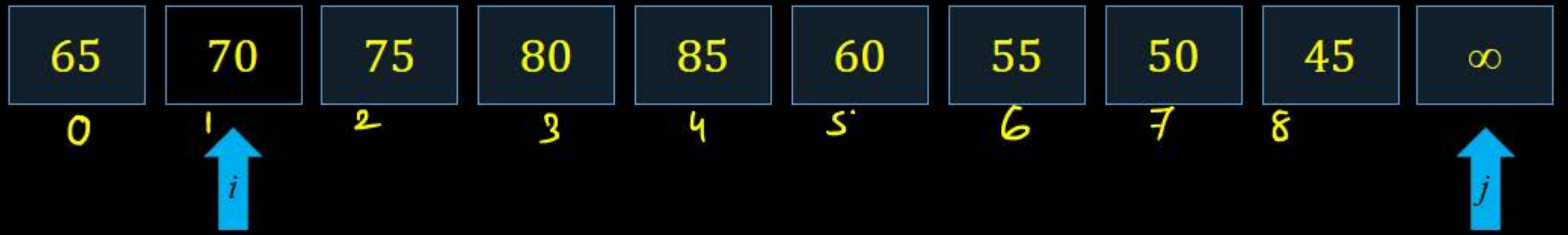
until $a[v] > a[i]$

repeat

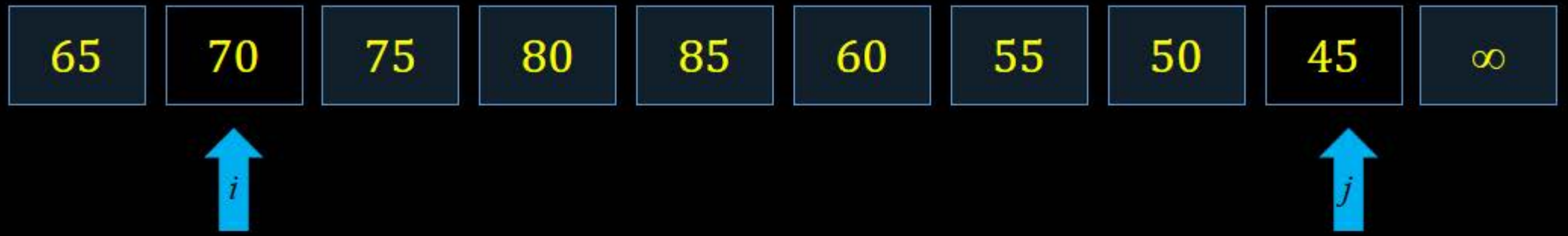
$j = j - 1$

until $a[v] < a[j]$

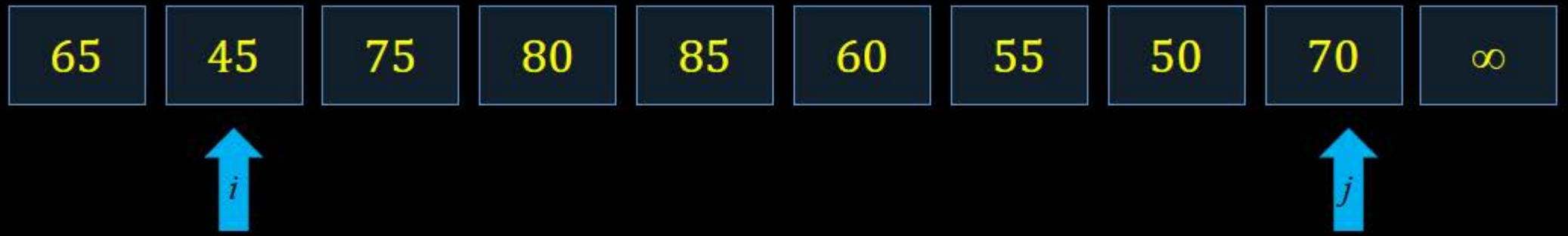
Hoare Partitioning



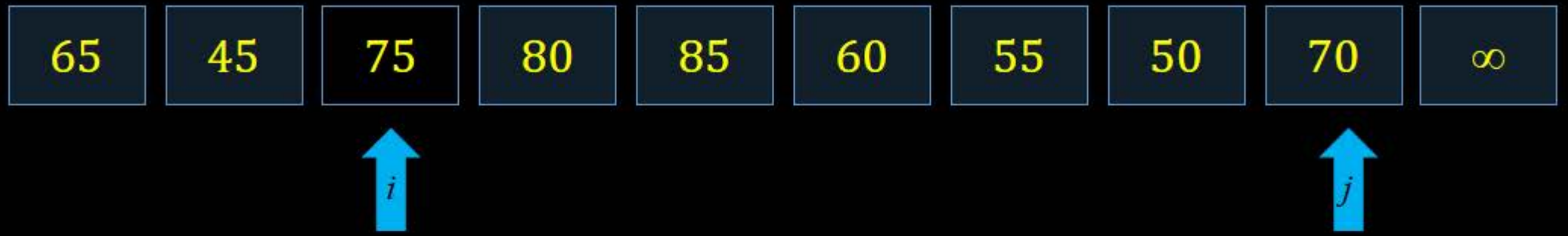
Hoare Partitioning



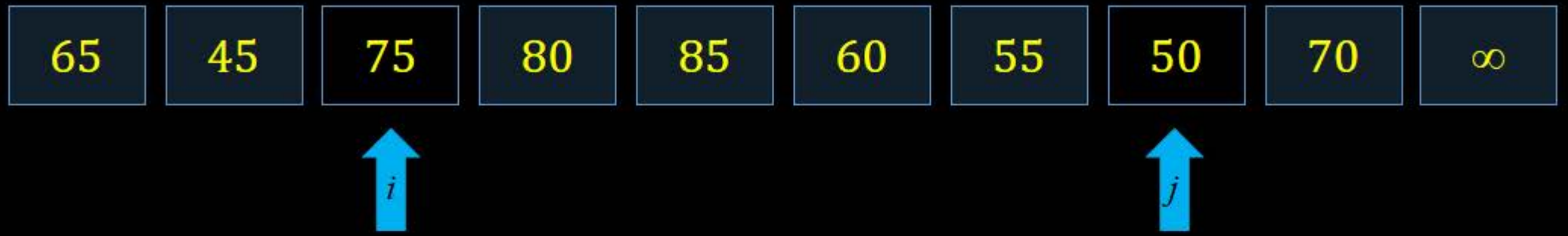
Hoare Partitioning



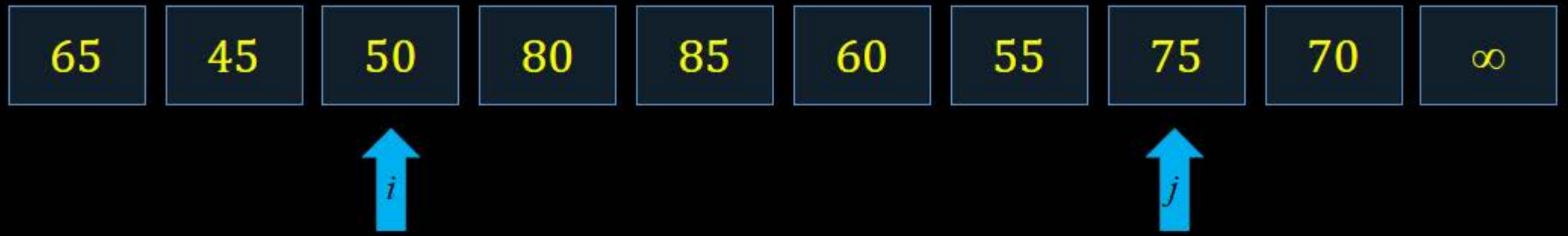
Hoare Partitioning



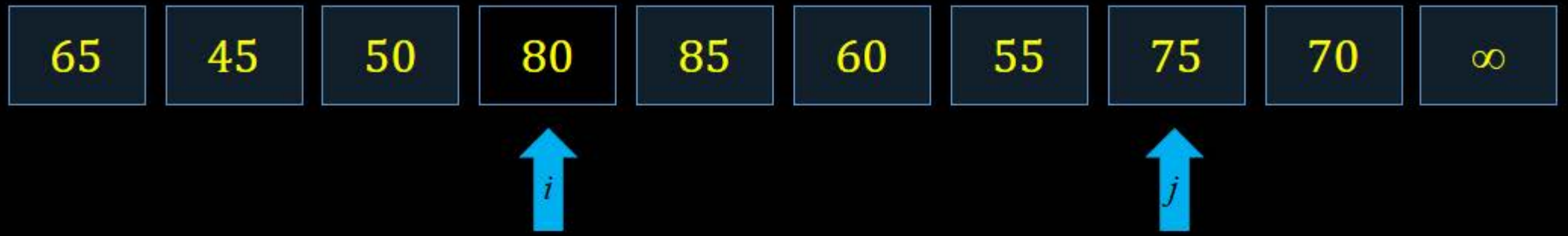
Hoare Partitioning



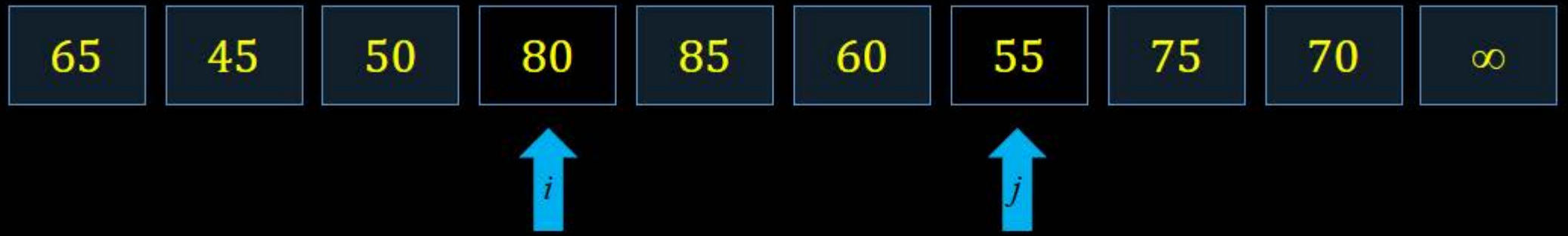
Hoare Partitioning



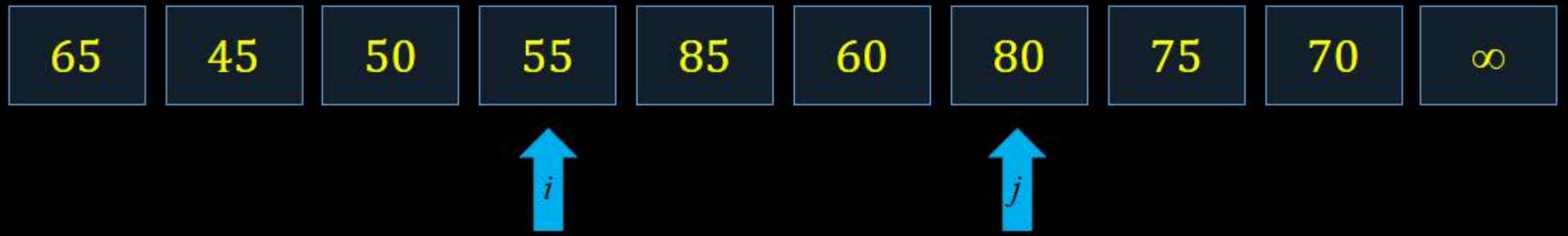
Hoare Partitioning



Hoare Partitioning



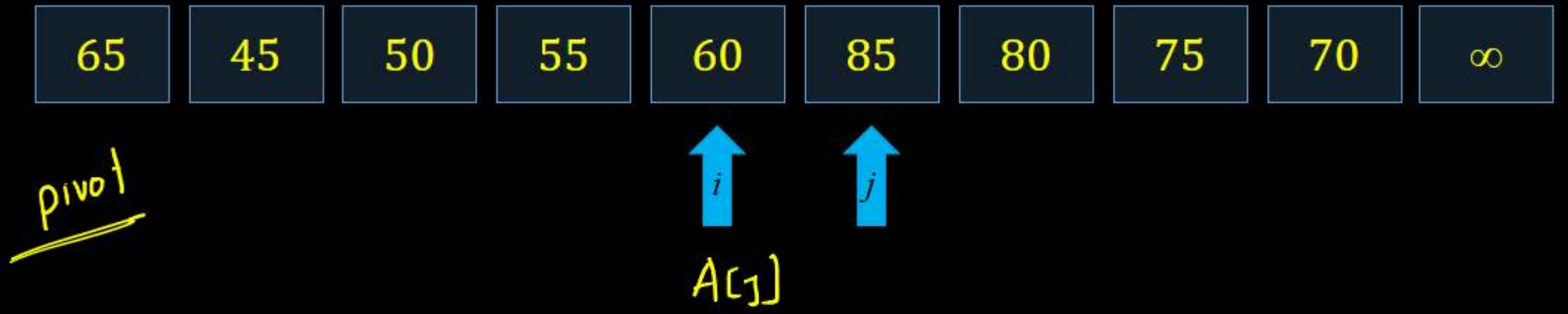
Hoare Partitioning



Hoare Partitioning

Hoare Partitioning

Hoare Partitioning



Hoare Partitioning

Hoare Partitioning

Hoare Partitioning

Hoare Partitioning

A[1 ... j-1]

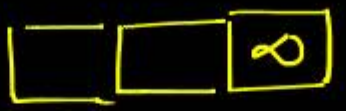
A[j] in its correct position

Gs (j+1, g)

5-1 = 4

The QuickSort Method

```
Algorithm QuickSort(p, q)
   if (p < q) then{
      j := Partition(a, p, q + 1);
      QuickSort(p, j - 1);
      QuickSort(j + 1, q);
   }
}
```



depending upon index j

The QuickSort Method

```
Algorithm QuickSort(p, q)
   if (p < q) then{
      j := Partition(a, p, q + 1);
      QuickSort(p, j - 1);
      QuickSort(j + l, q);
   }
}
```

The Partition Subroutine

```
Algorithm Partition(a, m, p){
v := a[m]; i := m; j := p;
repeat {
   repeat
      i := i + 1;
   until (a[i]  >= v);
   repeat
      j=j- 1;
   until (a[j]<= v);
   if (i < j) then Interchange(a,i,j);
} until (i >= j);
a[m] := a[j]; a[j] := v; return j;}
}
```

a[m] - pivot element

p - q+1

65	70	75	80	85	60	55	50	45	∞
60	45	50	55	60	85	80	75	70	
i	i	i	i	i	j	j	j	j	j

repeat until - repeat body of two unit condition becomes

Swap

return

The Partition Subroutine

Use the initial value of A[p] as the "pivot," in the sense that the keys are compared against it. Scan the keys A[p..q − 1] and rearranges them.

Hear In quick the ~~divided~~ deviding process depends upon Pivot.

Quick ( ) ← 0-elemul

Quick sort (n-1)

worst Case

1	4	3	5	2	←

0 ↑ (n-1)

3	4	1	2	5

Maxmin – division was very fixed

mid = $\frac{(i+j)}{2}$

what will be the size of partition