# Quick Sort

Divide And Conquer

Module 2

# Quick Sort

- Quick Sort uses Divide and Conquer Strategy.
- There are three steps:

1. **Divide:**
    - Splits the array into sub arrays.
    - Splitting of array is based on **pivot element**.
    - Each element in left sub array is less than and equal to middle (pivot) element.
    - Each element in right sub array is greater than the middle (pivot) element.

2. **Conquer:** Recursively sort the two sub arrays
3. **Combine:** Combine all sorted elements in a group to form a list of sorted elements.

# Example

**Step 1:**

Low High

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 50 | 30 | 10 | 90 | 80 | 20 | 40 | 70 |

i / **Pivot** j

**Step 2:** Increment i if A[i]<= Pivot and continue to increment it until element pointed by i is greater than A[Low]

High

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 50 | 30 | 10 | 90 | 80 | 20 | 40 | 70 |

Pivot i j

**Step 3:** Decrement j if A[j]> Pivot and continue to decrement it until element pointed by j is less than A[Low]

High

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 50 | 30 | 10 | 90 | 80 | 20 | 40 | 70 |

Pivot i j

# Example

**Step 4:**

As A[i] > A[Low], stop incrementing i

Low

High

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 50 | 30 | 10 | 90 | 80 | 20 | 40 | 70 |

i

Pivot

j

**Step 5:**

Increment i if A[i]<= Pivot and continue to increment it until element pointed by i is greater than A[Low]

High

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 50 | 30 | 10 | 90 | 80 | 20 | 40 | 70 |

Pivot

i

j

**Step 6:**

Decrement j if A[j]> Pivot and continue to decrement it until element pointed by j is less than A[Low]

High

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 50 | 30 | 10 | 90 | 80 | 20 | 40 | 70 |

Pivot

i

j

# Example

**Step 7:**

As **A[j] > A[Low]**, stop decrementing j

Low High

| 50 | 30 | 10 | **90** | 80 | 20 | **40** | 70 |
|---|---|---|---|---|---|---|---|

i j

Pivot

**Step 8:**

Since i and j cannot be further incremented and decremented, we will **swap A[i] and A[j]**

Low High

| 50 | 30 | 10 | **40** | 80 | 20 | **90** | 70 |
|---|---|---|---|---|---|---|---|

Pivot i j

**Step 9:**

Continue incrementing i and decrementing j until false conditions are obtained

Low High

| 50 | 30 | 10 | 40 | 80 | 20 | 90 | 70 |
|---|---|---|---|---|---|---|---|

i j

Pivot

# Example

**Step 7:**

Low High

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 50 | 30 | 10 | 40 | **80** | **20** | 90 | 70 |
| | | | | i | j | | |

Pivot

**Step 8:**

**swap A[i] and A[j]**

Low High

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 50 | 30 | 10 | 40 | **20** | **80** | 90 | 70 |
| | | | | i | j | | |

Pivot

**Step 9:**

Again Increment i and decrement j. **As soon as i > j, swap A[Low] and A[j]**

Low High

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **50** | 30 | 10 | 90 | **20** | 80 | 40 | 70 |
| | | | | j | i | | |

Pivot

# Example

swap A[Low] and A[j]

**Step 10:**

| Low | | | | Pivot | | | High |
|---|---|---|---|---|---|---|---|
| **20** | 30 | 10 | 40 | **50** | 80 | 90 | 70 |
| | | | | j | i | | |

**Step 11:**

| Low | | | | | | | High |
|---|---|---|---|---|---|---|---|
| **20** | 30 | 10 | 40 | **50** | 80 | 90 | 70 |
| i / Pivot | | | j | **mid** | i/Pivot | | j |

Left Sub Array

Right Sub Array

# Algorithm

```
Algorithm QuickSort(A , lb, ub)
{
if(lb < ub)……O(1)
{
Mid = partition(A,lb,ub)….O(n)(as we apply
partation on n elements)
QuickSort(A , lb, mid - 1)……T(n/2)
QuickSort(A , mid+1, ub)……T(n/2)
}

T(n)= 2T(n/2)+ n
```

```
Algorithm Partition(A, lb, ub)
{
        pivot = A[lb];
        start = lb;
        end =  ub;
        while(start < end)do
        {
                while(A[start] ≤ pivot)do
                { start++;}
                while(A[end] > pivot)do
                { end--;}
                if(start < end) then

        {swap(A[start],A[end])
        }}
        swap(A[lb],A[end])
        return end;
}
```

# Analysis

1. Best Case:
   - If array is partitioned at the mid
   - The Recurrence relation for quick sort for obtaining best case time complexity.

$$T(n) = T(n/2) + T(n/2) + cn \qquad \text{for } n > 1 \quad \ldots\ldots\ldots\ldots\ldots (1)$$
$$= 0 \qquad \text{for } n=1 \quad \ldots\ldots\ldots\ldots\ldots (2)$$

Using Master Theorem:

$$T(n) = 2 * T(n/2) + cn$$

$$\boxed{T(n) = \Theta(nlogn)}$$

Best case

$A = [n_1, n_2, n_3, n_4, n_5, n_6, n_7]$

pivot

pivot exactly at center after 1st iterations

$n$ — $n$

$n/2 \quad n/2$ — $\cancel{2} \times n/\cancel{2}$

$n/4 \quad n/4 \quad n/4 \quad n/4$ — $\cancel{4} \times n/\cancel{4}$

$T(1) \quad T(1) \quad T(1) \quad T(1)$ --- $n$

If we assume $k$ levels then cost

cost of subtree $= \underline{n \cdot k} = n \cdot \log n$

$T(1) = \dfrac{n}{2^k}$

$\therefore \dfrac{n}{2^k} = 1$

$n = 2^k, \; k = \log_2 n$

leaf nodes $= 2^k$

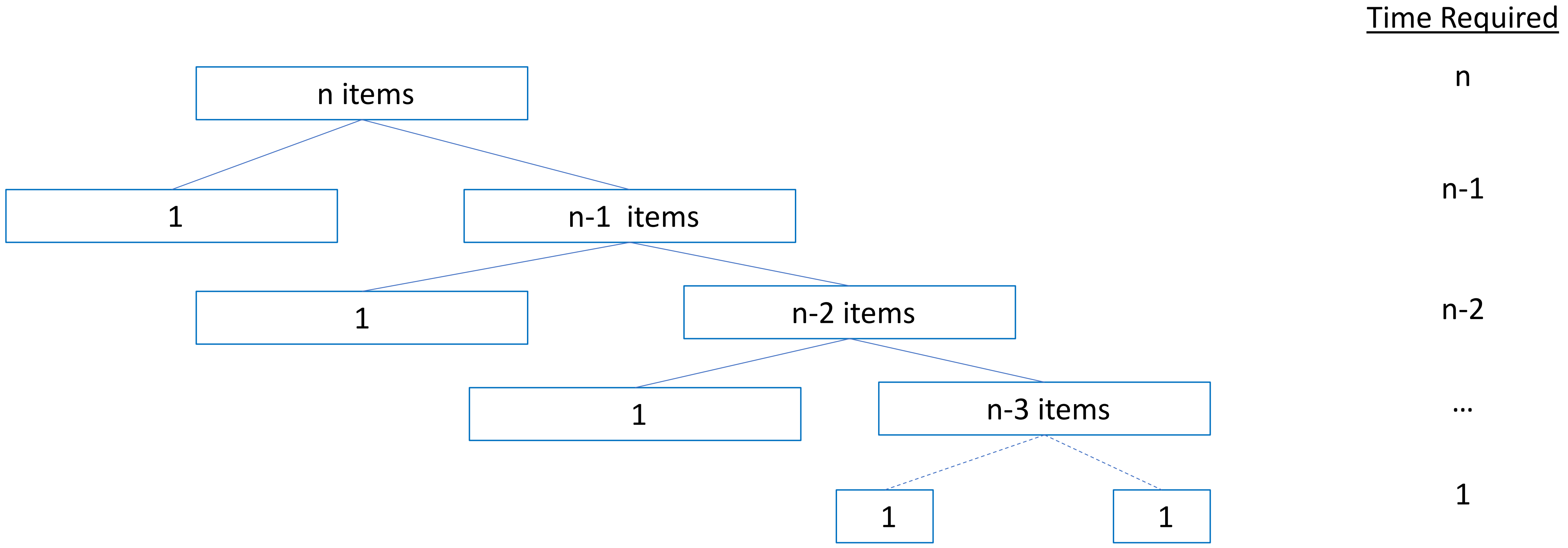
$= 2^{\log_2 n}$

Total cost $= n \log n \times$

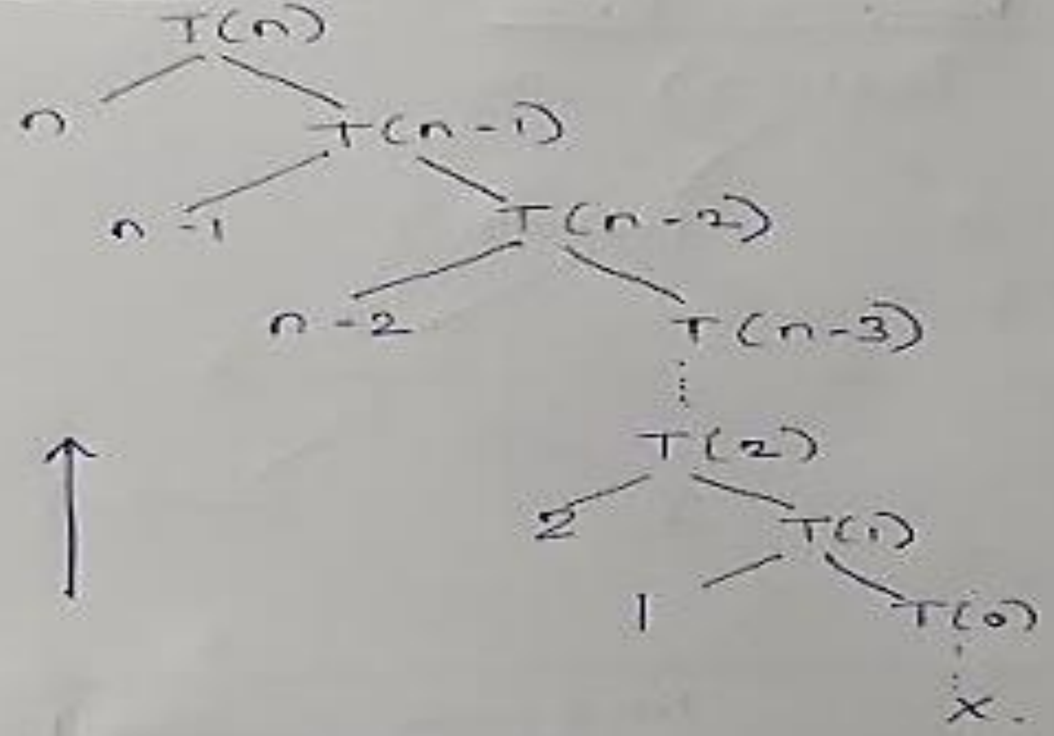
$= \underline{n \log n}$

# Analysis

2. Worst Case:
   - If pivot is a maximum or minimum of all the elements in the sorted list.
   - This can be graphically represented as follows



Recursion Tree




$$0+1+2+3 \cdots n-1+n = \frac{n(n+1)}{2} = n^2 = O(n^2)$$

# Analysis

## 2. Worst Case:

- If pivot is a maximum and minimum of all the elements in the sorted list.
- The Recurrence relation for quick sort for obtaining worst case time complexity according to the recursion tree(refer module1 notes for solving it).

$$T(n) = T(n-1) + n \quad \text{for } n > 1 \quad \ldots\ldots\ldots\ldots\ldots \text{①}$$

$$= 0 \quad \text{for } n = 1 \quad \ldots\ldots\ldots\ldots\ldots \text{②}$$

# Analysis

3. Average Case:
   - For any pivot position $i$; where $i \in \{0,1,2,3 \ldots n-1\}$
     - Time for partition an array: $cn$
     - Head and Tail sub-arrays contain $i$ and $n$-$1$-$i$ items.
     - So,

$$T(n) = T(i) + T(n-1-i) + cn$$

     - Average running time for sorting:

$$\boxed{T(n) = \frac{1}{n}\sum_{i=0}^{n-1}(T(i) + T(n-1-i)) + cn}$$

pivot Worst case, if array is already sorted.

| | | | | | | |
|---|---|---|---|---|---|---|
| 2 | 4 | 8 | 10 | 16 | 18 | 17 |

no left array. Subarray right

partition is done at beginning of list

1, 7 — $n$ comparison

2, 7 — $n-1$

3, 7 — $n-2$

4, 7

5, 7

6, 7

7, 7

2

1

if descending order partition will be done at end.

worst case time $\frac{n(n+1)}{2} = O(n^2)$

- If you have an array with 8 elements and the pivot is at the 3rd location, you can represent the time complexity using the quicksort recurrence relation. The typical recurrence relation for quicksort is as follows:
- $T(n) = \frac{1}{n}\sum_{i=0}^{n-1}(T(i) + T(n-1-i)) + cn$
- In this relation, *n* is the size of the array, and *i* is the position of the pivot after partitioning. If the pivot is at the 3rd location, then i=3.
- So for an array of 8 elements (*n*=8) and the pivot at the 3rd location (i=3), you can express the time complexity as:
- T(8)=T(3)+T(4)+Θ(8)
- This means that the time complexity of sorting an array of size 8 is equal to the time complexity of sorting the subarray on the left of the pivot (*T*(3)), plus the time complexity of sorting the subarray on the right of the pivot (*T*(4)), plus the time spent on the partitioning step (Θ(8)).