

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 $A[i] \leq \text{Pivot}$ and continue to increment it until element pointed by i is greater than $A[\text{Low}]$

High

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

Pivot

i

j

Step 3:

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

High

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

Pivot

i

j

Example

Step 4:

As $A[i] > A[\text{Low}]$, stop incrementing i

Low

High

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

i

j

Pivot

Step 5:

Increment i if $A[i] \leq \text{Pivot}$ and continue to increment it until element pointed by i is greater than $A[\text{Low}]$

High

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

i

j

Pivot

Step 6:

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

High

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

i

j

Pivot

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 |
|----|----|----|----|----|----|----|----|

i

j

Pivot

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

Step 10:

swap A[Low] and A[j]

| 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 | | | | | mid | i/Pivot | | |
| Left Sub Array | | | | | | Right Sub Array | | |
| j | | | | | | j | | |

Algorithm

```
Algorithm QuickSort(A[0...n], low, high)
{
    if(low < high) then
        mid ← partition(A[low...high])
        QuickSort(A[low...mid-1])
        QuickSort(A[mid+1...high])
}
```

```
Algorithm Partition(A[0...n], low, high)
{
    pivot ← A[low];
    i ← low;
    j ← high+1;
    while(i ≤ j) do
    {
        while(A[i] ≤ pivot) do
        { i++; }
        while(A[j] ≥ pivot) do
        { j--; }
        if(i ≤ j) then
            swap(A[i], A[j])
        }
        swap(A[low], A[j])
        return j;
    }
```


Analysis

1. Best Case:

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

$$\begin{array}{ll} T(n) = T(n/2) + T(n/2) + cn & \text{for } n > 1 \quad \dots\dots\dots \textcircled{1} \\ = 0 & \text{for } n = 1 \quad \dots\dots\dots \textcircled{2} \end{array}$$

Using Master Theorem:

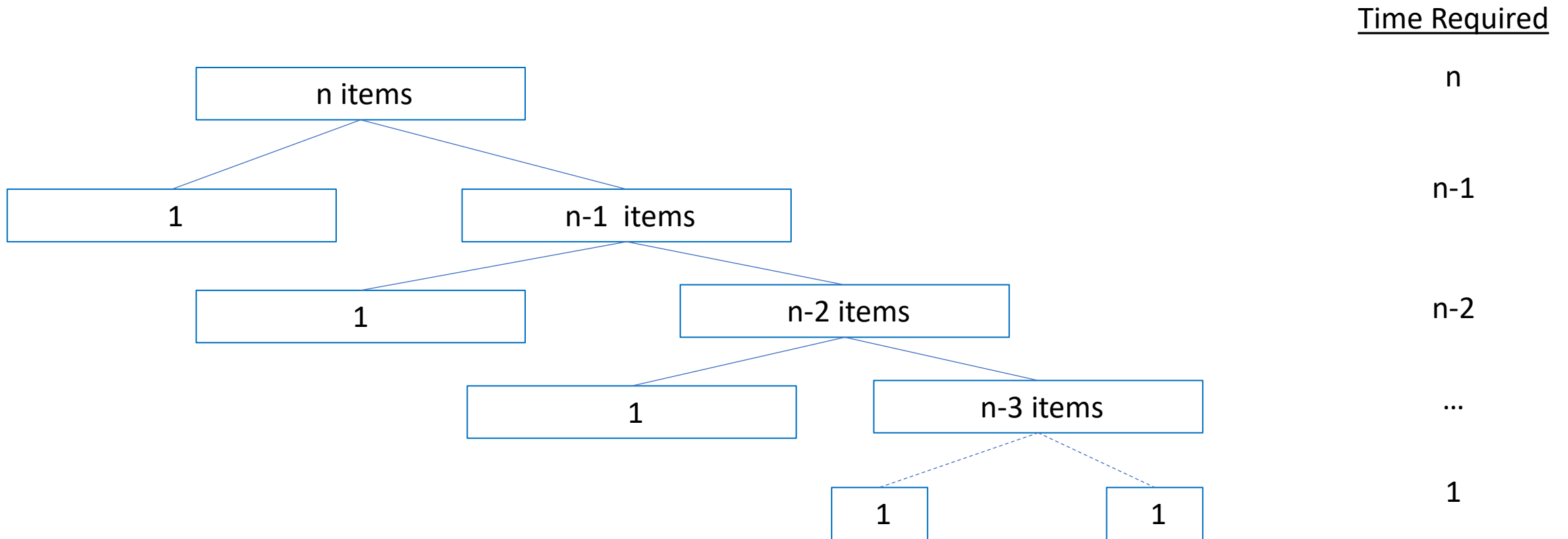
$$T(n) = 2 * T(n/2) + cn$$

$$T(n) = \Theta(n \log n)$$

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



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 best case time complexity.

$$\begin{aligned}T(n) &= T(n - 1) + cn \\ &= 0\end{aligned}$$

$$\begin{aligned}\text{for } n > 1 & \dots\dots\dots \textcircled{1} \\ \text{for } n = 1 & \dots\dots\dots \textcircled{2}\end{aligned}$$

Analysis

3. Average Case:

- For any pivot position i ; where $i \in \{0, 1, 2, 3 \dots n - 1\}$
 - 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:

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