Quick Sort

A Divide & Conquer Method

Quick Sort

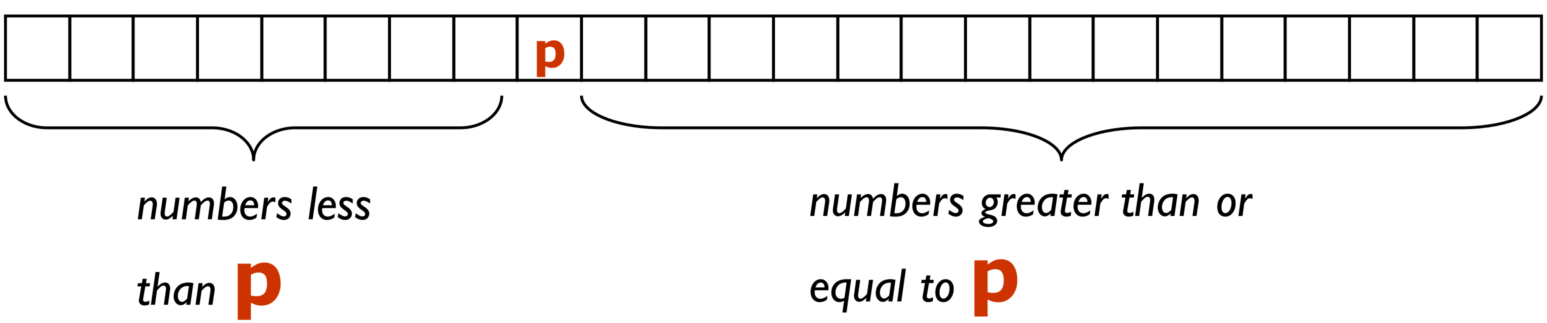
- Three steps in the divide and conquer strategy
  - Divide - Partition the original array A[p…..r] into two sub-arrays A[p……..q-1] and A[q+1………r] in such a way that each element of A[p……..q-1] is less than or equal to A[q],which in turn is less than or equal to each element of A[q+1…….r]
  - Conquer – The two sub arrays are then sorted by recursive calls
  - Combine- Since the sub arrays are sorted in place we cannot move the elements from the original array to some other array
  - q is the pivot element and the elements in sub array to the left of A[q] are less or equal to A[q] and elements in the sub array to the right of A[q] are equal or greater than A[q]

Quick sort

```
Quicksort(A[], lt, rt)
{
    if (lt < rt)
    {
        p = Partition(A[], lt, rt);
        Quicksort(A[], lt, p);
        Quicksort(A[], p+1, rt);
    }
}
```

Partitioning (Quick sort)

- A key step in the Quick sort algorithm is partitioning the array
  - We choose some (any) number **p** in the array to use as a pivot
  - We partition the array into three parts:



Partitioning (Quick sort) – Basic Idea

- Choose an array value (say, the first) to use as the pivot
- Starting from the left end, find the first element that is greater than or equal to the pivot
- Searching backward from the right end, find the first element that is less than the pivot
- Interchange (swap) these two elements
- Repeat, searching from where we left off, until done

Partitioning

- To partition a[left...right]:

1. **Set** p = a[left], l = left + 1, r = right;
2. **while** l < r, do
    - 2.1. **while** a[l] <= p, set l = l + 1
    - 2.2. **while** a[r] >= p, set r = r - 1
    - 2.3. **if** l < r, swap a[l] and a[r]
3. **Set** a[left] = a[r], a[r] = p
4. Return r
5. Terminate

Animation

10	16	8	12	15	6	3	9	5
10	16	8	12	15	6	3	9	5
10	5	8	12	15	6	3	9	16
10	5	8	12	15	6	3	9	16
10	5	8	9	15	6	3	12	16
10	5	8	9	15	6	3	12	16
10	5	8	9	3	6	15	12	16
10	5	8	9	3	6	15	12	16
6	5	8	9	3	10	15	12	16
6	5	8	9	3	10	15	12	16

6 5 8 9 3 10 15 12 16

6 5 8 9 3

6 5 3 9 8

6 5 3 9 8

3 5 6 9 8

3 5 6 9 8

3 5

9 8

3 5

8 9

15 12 16

15 12 16

15 12 16

12 15 16

12 15 16

Worst Case Call Tree (N=4)

S =[1,3,5,7]

Q(1,4)

Left=1, pivotitem = 1, Right =4

1

Q(2,4)

Left =2,pivotItem=3

S =[3,5,7]

3

Q(3,4)

pivotItem = 5, Left = 3

S =[5,7]

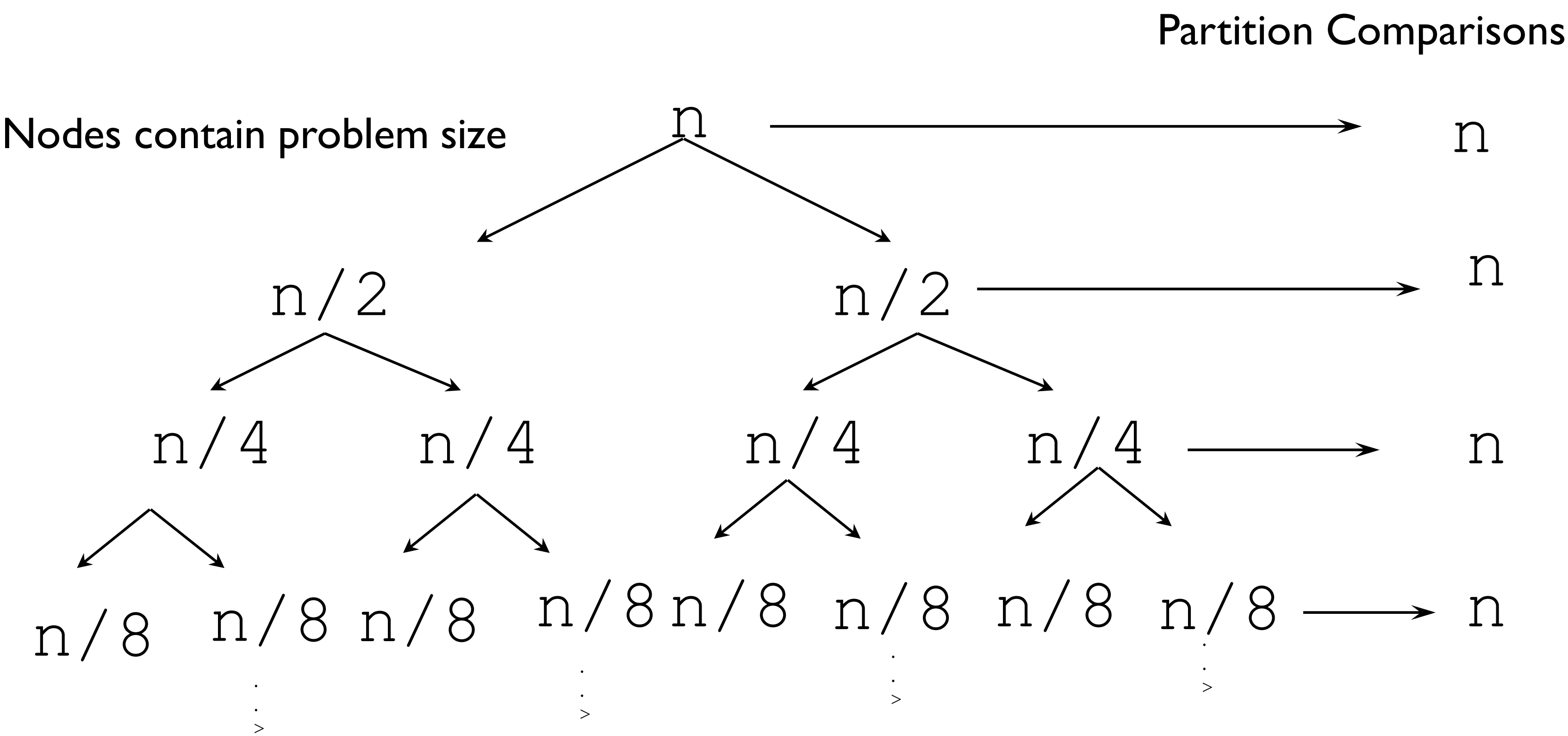
5

Q(4,4)

S =[7]

7

Recursion Tree for Best Case



Sum $=\Theta(n \lg n)$

Features -Quick Sort

- Performs very well in virtual memory environment
- Needs fewer resources during execution
- Running time depends on whether the partitioning is balanced or unbalanced
- When the sub list is balanced it is the best case and worst case if it is vice –versa
- If the sub list is balanced then it runs as fast as merge sort
- If the sub list is unbalanced then it runs as slow as insertion sort

Complexity Analysis- Quick Sort

- Worst Case
  - Partitioning is unbalanced and partition procedure produces two sub arrays , one with n-1 elements and the other containing zero elements.
  - The recurrence can be written as
    - $T(n)=T(n-1)+T(0)+ \theta(n)$
    - $T(n)=T(n-1)+ \theta(n)$ since $T(0)= \theta(1)$
    - Solving using the iteration method we get
      - $T(n)= \theta(n^2)$
- Best Case
    - Partitioning is balanced and it produces two sub arrays containing not more than n/2 elements.
    - The recurrence for such a case can be written as
      - $T(n)=T(n/2) +T(n/2)+\theta(n)$
      - $T(n)=2T(n/2)+ \theta(n)$
      - Applying master theorem we get
        - $T(n)= \theta(n\log n)$

Contd…

- Average Case
  - $T(n)= O(nlogn)$
- An algorithm is called stable if it preserves the order of elements in both input and output
- Insertion sort , bubble sort , shell sort , merge sort are stable
- Selection sort and quick sort are unstable algorithms


Thank you!