Design & Analysis of Algorithms

[Divide & Conquer]
QuickSort & Exponentiation/Power

Soharab Hossain Shaikh
***BML Munjal* University**

1

Quick Sort: D&C Approach

- Suppose we are given an unsorted array A[$p..r$].
- The three steps for the design of the algorithm:
  - **Divide**: Partition array $A[p..r]$ into two subarrays $A[p..q]$ and $A[q+1..r]$ such that each element of $A[p..q]$ is less than or equal to each element of $A[q+1..r]$. The index q is computed by a partitioning algorithm known as SPLIT algorithm.
  - **Conquer**: The two subarrays $A[p..q]$ and $A[q+1..r]$ are sorted by recursive calls to Quicksort.
  - **Combine**: Since the subarrays are sorted in place, no work is needed to combine them: the entire array $A[p..r]$ is now sorted.

2

SPLIT

- **SPLIT algorithm**

Algorithm SPLIT
Input $A[low..high]$

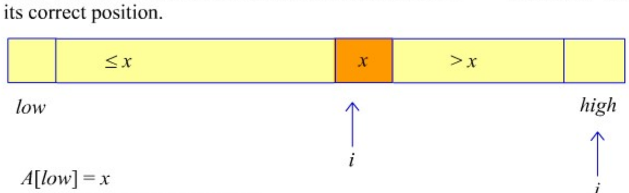
```
i ← low
x ← A[low]
for j ← low + 1 to high
    if A[j] ≤ x then
        i ← i+1
        if i ≠ j then interchange A[i] and A[j]
    end if
end for
interchange A[low] and A[i]
w ← i
return A and w
```

3

SPLIT : Illustration

➢ Some Observations:

After partitioning an array A, using the first element $x \in A$ as a pivot, x will be in its correct position.



$A[low] = x$
$A[k] \le x$ for all k, $low \le k \le i$
$A[k] > x$ for all k, $i < k \le j$.

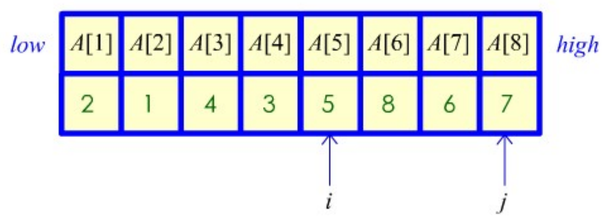
- The time complexity of the SPLIT algorithm is $\Theta(n)$.

- The space complexity of the SPLIT algorithm is $\Theta(1)$.

4

SPLIT : Illustration

- Example



Quick Sort Algorithm

- **Algorithm Quicksort**

Algorithm Quicksort
Input: An array $A[1 \,..\, n]$ of n elements
Output: The elements in A sorted in nondecreasing order.

Quicksort (A,1, n)

Procedure Quicksort (A,1, n)

if $low < high$ **then**
    SPLIT($A[low \,..\, high]$,w)   { where w is the new position of $A[low]$ }
    Quicksort(A, low, w-1)
    Quicksort(A, w+1 , $high$)
endif

Quick Sort Analysis

- **Observations:**

  - The call SPLIT in step 2 is the divide step.
  - Both calls to Quicksort procedure are part of conquer step.
  - No combine step needed.

Algorithm Quicksort
Input: An array $A[1 .. n]$ of n elements
Output: The elements in A sorted in nondecreasing order.

1. Quicksort (A,1, n)

Procedure Quicksort (A,1, n)

1. **if** $low < high$ **then**
2.   SPLIT($A[low .. high]$,w) { where w is the new position of $A[low]$ }
3.   Quicksort(A, low, w-1)
4.   Quicksort(A, w, $high$)
5. **endif**

7

Quick Sort Example

- **Example**

4	6	3	1	8	7	2	5	
2	3	1	**4**	8	7	6	5	**1st Call**
1	**2**	3						**2nd Call**
1		**3**		5	7	6	**8**	**3rd Call/4th Call/5th Call**
				5	7	6		**6th Call**
					6	**7**		**7th Call/8th Call**
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**Sorted Array**

8

Quick Sort Complexity

- **Time Complexity**
  - $\Theta(n^2)$ in the worst case
  - $\Theta(n \log n)$ in the average case

9

Quick Sort Analysis (cont..)

- **Worst Case Analysis**
  - Consider a scenario that in every call to SPLIT, $A[low]$ is the lowest among all other elements in A. Therefore,
    - The algorithm SPLIT will return $w = low$. Thus, only one of the recursive calls to Quicksort, i.e. the call at Step 4, is effective.
    - The call at step 3 will have no cost.
  - In fact the procedure Quicksort has the following calls:
    Quicksort(A,1,n), Quicksort(A, 2, n), Quicksort(A,3, n), … , Quicksort(A, n, n).
  - This means we are in turn calling SPLIT procedure as the following calls:
    SPLIT(A[1..n], w), SPLIT(A[2..n], w), SPLIT(A[3..n], w), …, SPLIT(A[n..n], w).
  - Every call to SPLIT(A[1 .. j], w) takes j-1 comparisons. Thus, the total cost is

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

Thus, the worst case analysis of Quicksort is $\Theta(n^2)$.

10

Pivot Selection Methods

1. Select the first or the last element of the array as pivot (as done already).

2. Randomized pivot selection - **Randomized Split/Partition**

```
RandomizedSplit(A, low, high)
{
  idx = RandomIndexSelection(low, high)
  swap (A[idx], A[low])
  Split(A[low..high], w)
}
```

3. **Median of Three** (median of the first, mid and the last elements of the array – A[low], A[mid], A[high] where mid=low+(high-low)/2)

11

Quick Sort Analysis (cont..)

- **Average-case Analysis**
  - For the average case analysis we need the few assumptions:
  - All elements in array A are distinct.
  - Probability that any element of A will be picked as the pivot is $1/n$.

  - Algorithm Quicksort contains:
  - A call to procedure SPLIT that takes $n-1$ comparisons, and
  - Two calls: Quicksort(A, 1, $w-1$) and Quicksort(A, $w+1$, n).

  - The recurrence relation obtained is as follows:

$$C(n) = n - 1 + \frac{1}{n}\sum_{w=1}^{n}\left(C(w-1) + C(n-w)\right).$$

This can be simplified to $C(n) = n - 1 + \frac{2}{n}\sum_{w=1}^{n} C(w-1)$,

as $\sum_{w=1}^{n} C(n-w) = C(n-1) + C(n-2) + \ldots + C(0) = \sum_{w=1}^{n} C(w-1)$.

12

Quick Sort Analysis (cont..)

Multiplying the equation $C(n) = n - 1 + \frac{2}{n}\sum_{w=1}^{n} C(w-1)$, by n yields:

$$nC(n) = n(n-1) + 2\sum_{w=1}^{n} C(w-1).$$

If we replace n by $n-1$ in the above equation, we obtain

$$(n-1)C(n-1) = (n-1)(n-2) + 2\sum_{w=1}^{n-1} C(w-1).$$

Subtracting the above two equations and rearranging the terms, we obtain

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

Now we change to a new variable D, by letting

$$D(n) = \frac{C(n)}{n+1}.$$

13

Quick Sort Analysis (cont..)

In terms of the new variable D, we have the new formulation as follows:

$$D(n) = D(n-1) + \frac{2(n-1)}{n(n+1)}, \text{ where } D(1) = 0$$

Using expansion, we can easily see that

$$D(n) = 2\sum_{j=1}^{n} \frac{j-1}{j(j+1)}.$$

We simplify $D(n)$ as follows:

$$2\sum_{j=1}^{n} \frac{j-1}{j(j+1)} = 2\sum_{j=1}^{n} \frac{2}{(j+1)} - 2\sum_{j=1}^{n} \frac{1}{j} \text{ (using partial fractions)}$$

$$= 4\sum_{j=2}^{n+1} \frac{1}{j} - 2\sum_{j=1}^{n} \frac{1}{j} \text{ (replacing } j\text{+1 by } j \text{ and changing the summation indices accordingly)}$$

$$= 2\sum_{j=2}^{n} \frac{1}{j} + \frac{4}{n+1} - 2 = 2\sum_{j=1}^{n} \frac{1}{j} + \frac{4}{n+1} - 4 = 2\sum_{j=1}^{n} \frac{1}{j} - \frac{4n}{n+1}.$$

14

Quick Sort Analysis (cont..)

$$2\sum_{j=1}^{n}\frac{j-1}{j(j+1)} = 2\sum_{j=1}^{n}\frac{1}{j} - \frac{4n}{n+1}$$

$$= 2\ln n - \Theta(1) \quad (\text{since } \sum_{j=1}^{n}\frac{1}{j} = \ln n)$$

$$= \frac{2}{\log e}\log n - \Theta(1)$$

$$\approx 1.39 \log n.$$

log 2 / log e = 0.693

So, log e / log 2 = 1/0.693 = 1.44

So, 2/(log e / log 2) = 2/1.44 = 1.3888 ≈ 1.39

Consequently,

$$C(n) = (n+1)D(n) \approx 1.39\, n \log n$$

So the average case running time complexity of algorithm Quicksort is $\Theta(n \log n)$.

On average, Quicksort makes 39% more computation than the theoretical lower bound for comparison-based sorting methods.

15

Exponentiation/Power

16

Exponentiation/Power

Find the x^n for x>0 and n is non-negative integer.

```
Iterative_Power (x, n)
{
   res = x
   For i=2 to n
     res = res *x
  return (res)
}
```

Time complexity = O(n)

17

Recursive Exponentiation

Find the x^n for x>0 and n is non-negative integer.

Recursive formulation : $x^n = x * x^{n-1}$ *if n>0*

$= 1$ *if n=0*

```
Rec_Power1(x, n)
{
   if (n==0)
    return (1)
   else
    return (x * Rec_Power1(x, n-1))
}
```

Time complexity:

T(0) =1;

T(n)= T(n-1) + C = T(n-2) + 2C … =T(n-k) + k.C = T(0) + k.C = 1+ k.C

= 1+ n.C [as n-k=0]

= O(n)

18

Recursive Exponentiation

Find the $\mathbf{x^n}$ for $x>0$ and n is non-negative integer.

Recursive formulation : $\mathbf{x^n = x^{n/2} * x^{n/2}}$ *if n is even*

$= \mathbf{x * x^{(n-1)}}$ *if n is odd*

```
Rec_Power2(x, n)
{
   if (n==0)
     return (1)
   else
     if(n%2==0)
      {
        y = Rec_Power2(x, n/2)
        return (y * y)
      }
   else
      return (x * Rec_Power2(x, n-1))
}
```

19

Recursive Exponentiation

Find the $\mathbf{x^n}$ for $x>0$ and n is non-negative integer.

Recursive formulation : $\mathbf{x^n = x^{n/2} * x^{n/2}}$ *if n is even 0*

$= \mathbf{x * x^{(n-1)}}$ *if n is odd*

Time complexity:

T(0) =1;

T(n)= T(n/2) + c1 if n is even = = => O(logn) [like Binary Search]

= T(n-1) + c2 if n is odd

T(1) = T(0) + c2 = 1+c2

If initially n is odd then, assuming $n= 2^k +1$

$T(n) = T(n-1) + c2 = T([n-1]/2) + c2$

$= T(n/2) + c1 + c2$ ***[taking floor integral value of n, now problem size n is*** 2^k***]***

$= T(n/4) + c1 + c$ $[c=c1+c2]$

.... $= T(n/2^k) + k.c1 + c = T(1) + k.c1 + c = 1+c2 + c + k.c1$

$= C' + c1. \log n = O(\log n)$ ***[as k = logn]***

20

End of Lecture