Data Structures and Algorithms

Mohammad GANJTABESH

Department of Computer Science,
School of Mathematics, Statistics and Computer Science,
University of Tehran,
Tehran, Iran.

mgtabesh@ut.ac.ir

Dept. of Computer Science
University of Tehran

Sorting Algorithms

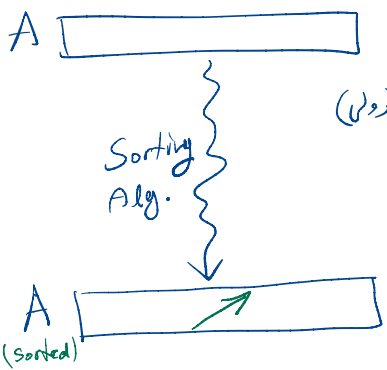
Data Structures and Algorithms
Undergraduate course

Mohammad GANJTABESH
mgtabesh@ut.ac.ir

Dept. of Computer Science
University of Tehran

Motivations

- ترجیح بر این است که داده ها همواره مرتب باشند.



آرایه‌ای از داده

ترتیب صعودی (نزولی)

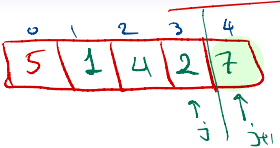
- مهمترین کاربرد: جستجوی داده

الگوریتم های مرتب سازی

برپایه مقایسه

برپایه شمارش

Bubble Sort




```
BubbleSort (A, n) {
    len ← n
    for (i ← 0 to n-2) {
        for (j ← 0 to len-2) {
            if (A[j] > A[j+1]) {
                swap (A[j], A[j+1])
            }
        }
        len ← len - 1
    }
}
```

O(n)

O(n)

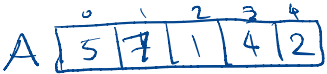
Time Complexity ⇒ $O(n^2)$

Selection Sort

```
SelectionSort(A, n){
    for(i ← 0 to n-2){
        i_min ← i
        for(j ← i+1 to n-1){
            if(A[j] < A[i_min]){
                i_min ← j
            }
        }
        swap(A[i], A[i_min])
    }
}
```

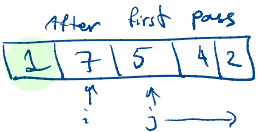
O(n) — outer loop; O(n) — inner loop

Time Complexity = $O(n^2)$



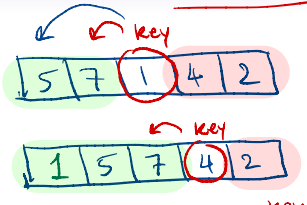
index_min = 2

swap(A[0], A[index_min])



Insertion Sort

```
InsertionSort (A, n) {
    for ( i ← 1 to n-1) {
        key ← A[i]
        pos ← i
        while ( pos > 0 & A[pos-1] > key) {
            A[pos] ← A[pos-1]
            pos ← pos - 1
        }
        A[pos] ← key
    }
}
```



Best Case:

O(n)

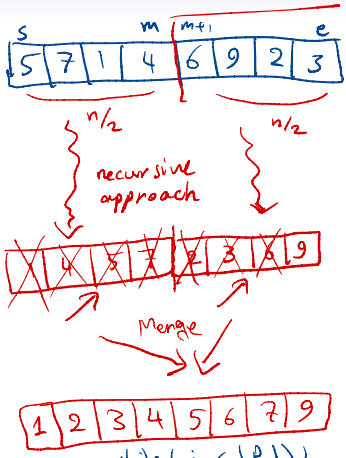
Worse Case:

$O(n^2)$

Merge Sort

```
MergeSort (A, s, e) {
    if (s<e) {
        m ← ⌊(s+e)/2⌋
        MergeSort (A, s, m)
        MergeSort (A, m+1, e)
        Merge (A, s, m, e)
    }
}
```

```
Merge (A, s, m, e) {
    L ← A[s, m]
    R ← A[m+1, e]
    i ← 0
    j ← 0
    k ← 0
    while (i<|L| & j<|R|) {
        if (L[i] ≤ R[j]) {
            A[k] ← L[i]
            i ← i+1
        }
        else {
            A[k] ← R[j]
            j ← j+1
        }
        k ← k+1
    } // for while
    while (i<|L|) {
        A[k] ← L[i]
        i ← i+1
        k ← k+1
    }
    while (j<|R|) {
        A[k] ← R[j]
        j ← j+1
        k ← k+1
    } // for Merge
}
```



Merge Sort (Analysis)؟

فرض کنید طول آرایه داده شده n باشد.

$$T(n) = \begin{cases} O(1) & \text{if } n=1 \\ 2T(n/2) + O(n) & \text{o.w.} \end{cases}$$

$$T(n) \leq 2T(n/2) + c.n$$

$$\Rightarrow T(2^k) \leq 2T(2^{k-1}) + c.2^k$$

$$n = 2^k$$
$$n/2 = 2^{k-1}$$
$$k = \log_2 n$$

$$\leq 2\left[2T(2^{k-2}) + c.2^{k-1}\right] + c.2^k$$
$$= 2^2 T(2^{k-2}) + 2.c.2^k$$
$$\leq 2^2\left[2T(2^{k-3}) + c.2^{k-2}\right] + 2c2^k$$
$$= 2^3 T(2^{k-3}) + 3.c.2^k$$
$$\vdots$$
$$\leq 2^k T(1) + k.c.2^k = 2^k.c' + k.c.2^k = O(n.\log_2 n)$$

روش‌های حل روابط بازگشتی

1- روش جایگذاری

2- درخت بازگشت

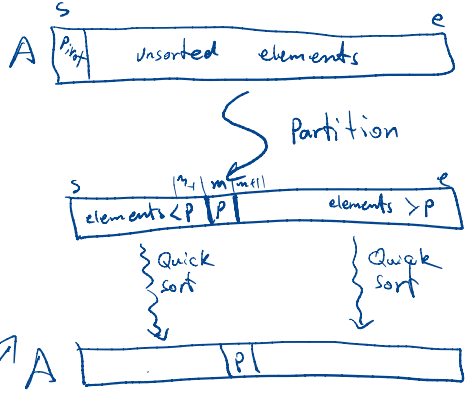
3- توابع مولد

4- قضیه اصلی (Master theorem)

Quick Sort

$$T(n) = \begin{cases} O(1) & \text{if } n=1 \\ ? \end{cases}$$

```
QuickSort (A, s, e){
    if (s < e) {
        m ← Partition (A, s, e)
        QuickSort (A, s, m-1)
        QuickSort (A, m+1, e)
    }
}
```



Time complexity:

Best: $T(n) = 2T(n/2) + O(n) = O(n \log_2 n)$ ✓

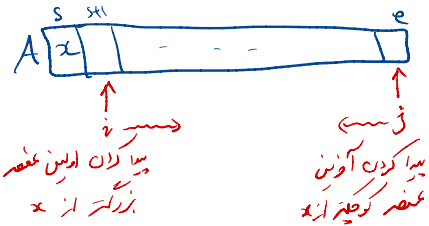
worse: $T(n) = T(0) + T(n-1) + O(n) = O(n^2)$ ✗

Average: $O(n \log_2 n)$ ✓

using median based on k-select Algorithm

Quick Sort

```
Partition(A, s, e){
  // based on pivot ← first element
  x ← A[s]
  i ← s+1
  j ← e
  repeat {
      while (A[i] < x)   i ← i+1
      while (A[j] > x)   j ← j-1
      if (i < j){
          swap (A[i], A[j])
      }
  } until (i < j)
  swap (A[s], A[j])
} return (j)
```



Time complexity:

$O(n)$

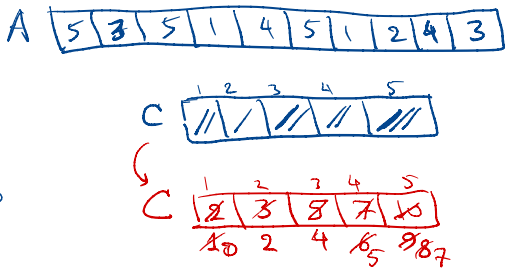
Counting Sort

$0 \le$ کوچک‌ترین $< k$

تعداد اعداد $= n$

$k \ll n$ ✓

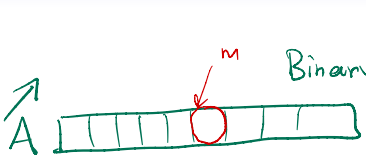
```
CountingSort(A){
  // C is an array of len. k
  for(i ← 0 to k)   C[i] ← 0
  for(j ← 0 to n)
      C[A[j]]++
  for(i ← 1 to k)
      C[i] ← C[i] + C[i-1]
  for(j ← 0 to n){
      B[C[A[j]] - 1] ← A[j]
      C[A[j]]--
  }
}
```



Time Complexity:

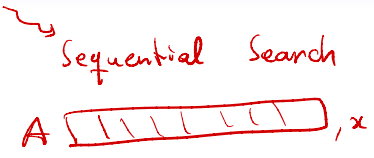
$O(n+k)$

Searching: Sorted vs. Unsorted Arrays




```
BinarySearch (A, s, e, x){
    m ← ⌊(s+e)/2⌋
    if (A[m] = x)  return (m)
    else if (x < A[m])
        return (BinarySearch (A, s, m-1, x))
    else
        return (BinarySearch (A, m+1, e, x))
}
```

Time Complexity : $O(\log_2 n)$




```
SequentialSearch(A,n,x){
    for (i ← 0 to n-1){
        if (A[i] = x)
            return (i)
    }
    return (-1)
}
```

Time Complexity : $O(n)$