Advanced Analysis of Algorithms

Department of Computer Science
Swat College of Science & Technology

CS Course : Advanced Analysis of Algorithms
Course Instructor : Muzammil Khan

Discussion Class

- We have discussion class next week, on
  - Data structures
    - Arrays
    - Stacks
    - Queues
    - Records and pointers
    - Lists
    - Graphs
    - Trees
    - Associative Tables
    - Etc…

Chapter 5

Sorting Algorithms
(Elementary Sort Algorithms)

Other Resources

- Animated Sorting Examples
  - http://www.ee.ryerson.ca/~courses/coe428/intro/introduction.html

Sorting Algorithms - Classification

- The sorting algorithms are classified into two categories
  - On the basis of *underlying procedure* used
- Sorting by Comparison
  - These are based on comparison of keys
    - The method has general applicability
    - Examples are selection sort, quick sort
- Sorting by Counting
  - These depend on the characteristics of individual data items
    - The sort method has limited application
    - Examples include radix sort, bucket sort,

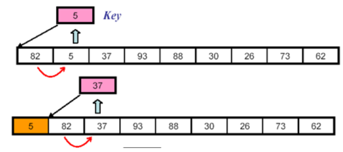
Sorting by Comparison

- The *Sorting by Comparison* method sorts input
  - By comparing pairs of keys in the input.
- Elementary Algorithms
  - Are *inefficient* but ***easy to implement***
  - Their running times are $\theta(n^2)$
  - Common algorithms are
    - *Insertion sort, Exchange sort, Selection sort*
- Advanced Algorithms
  - Advanced algorithms are ***efficient***
  - Implementations are usually based on ***recursive function calls***
  - Their running times are $\theta(n \lg n)$
  - The typical advanced algorithms include
    - *Merge sort, Quick sort, Heap sort*

Insertion Sort

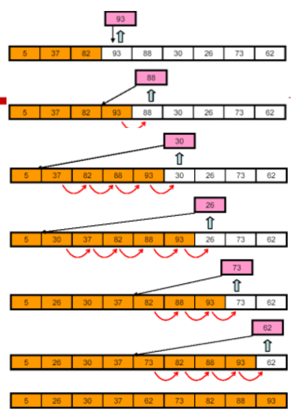
Insertion Sort

- The key is compared with the successive elements on the left
  - Until a smaller element is found
- Key is inserted
  - By shifting all the elements to right



Insertion Sort

- Illustration



Insertion Sort

The **INSERT-SORT** method sorts an input array *A[1…n]* using ***insertion sort*** method.

```
INSERT-SORT(A)
n ← length[A]                          ► n holds array size
for j ← 2 to n  do                     ► Examine elements A[2]….A[n]
    key ← A[j]                         ► Select key
    i ← j – 1
    while i > 0 and A[i] > key  do     ► Scan backwards for an element smaller than key
        A[i+1] ← A[i]                  ► Shift toward right to create space for insertion of key
        i ← i-1
    A[i+1] ← key                       ► insert key in the vacant place
```

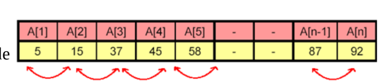

Insertion Sort Analysis

INSERT-SORT(A)	Cost	Times
$n \leftarrow length[A]$	C_1	1
for $j \leftarrow 2$ *to* n ***do***	C_2	n
$key \leftarrow A[j]$	C_3	n – 1
$i \leftarrow j - 1$	C_4	n – 1
while $i > 0$ *and* $A[i] > key$ ***do***	C_5	$\sum_{j=2}^{n} t_j$
$A[i+1] \leftarrow A[i]$	C_6	$\sum_{j=2}^{n} (t_j - 1)$
$i \leftarrow i-1$	C_7	$\sum_{j=2}^{n} (t_j - 1)$
$A[i+1] \leftarrow key$	C_8	n – 1

$$T(n) = c_1 + c_2 n + c_3(n-1) + c_4(n-1) + c_5\sum_{j=2}^{n} t_j + c_6\sum_{j=2}^{n}(t_j - 1) + c_7\sum_{j=2}^{n}(t_j - 1) + c_8(n-1)$$

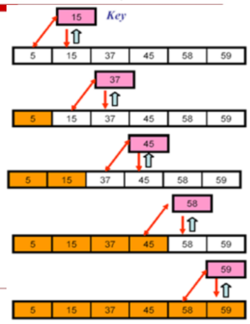
Insertion Sort (Best Case Scenario)

- The best case occur when
  - The input is *Presorted*
- The key is written back
  - After comparison
    - That is,
    - There is no moves / shifting to right of the elements
- Illustration
  - Next slide



Insertion Sort (Best Case Scenario) (Cont...)

- If A[2] is selected as *Key*
  - *One* comparison is done
- If A[3] is selected as *Key*
  - *One* comparison is done
- If A[4] is selected as *Key*
  - *One* comparison is done
- . And so on…
- If A[n] is selected as *Key*
  - *One* comparison is done
- Thus, ***n-1*** comparison are done
- Best case running of insertion sort
  - θ ***(n-1)*** $= \theta$ ***(n)***



Insertion Sort (Worst Case Scenario)

- The worst case occur when
  - The input is in *reverse sorted*
- The key is inserted in the left most position
  - Maximum number of comparison
    - That is,
    - Maximum number of moves / shifting to right of the elements involved
- Illustration
  - Next slide

A[1]	A[2]	A[3]	A[4]	A[5]	-	-	A[n-1]	A[n]
95	85	66	55	43	-	-	10	8

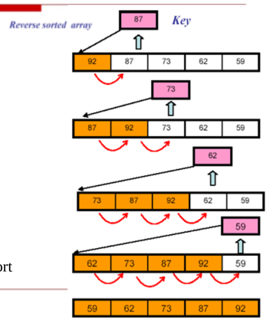
Insertion Sort (Worst Case Scenario) (Cont...)

- If A[2] is selected as *Key*
  - *One* comparison is done
- If A[3] is selected as *Key*
  - *Two* comparison is done
- If A[4] is selected as *Key*
  - *Three* comparison is done
- . And so on…
- If A[n] is selected as *Key*
  - ***n-1*** comparison is done

Thus, $T_w(n) = c[1+2+3+\ldots+(n-2)+(n-1)]$

$= c.\ n(n-1)/2 = \theta\ (n^2)$

- Worst case running of insertion sort
  - $\theta\ (n^2)$



Insertion Sort (Average Case Scenario)

- In order to find average case running time
  - Consider the *expected cost* of inserting an element in *subarray of k elements*
    - Using probabilistic analysis

A[1]	A[2	A[3]	A[4]	A[5]	-	A[k-1]	A[k]	A[k+1]	A[k2]	..	..	A[n]

- Insertion sort procedure looks for
  - The position of the element just *smaller than the key*
  - The insertion of requisite element has equal probability of being in any of the k locations
    - That is ***1 / k***

Insertion Sort (Average Case Scenario)

- Each *insertion of key cause one shift* to the right
  - Assuming that ***c*** is the *unit cost*
  - The following table summarizes
    - The number of shift operation &
    - Associated cost

Location of element smaller than *key*	probability	# of shifts operations	Cost
k	*1/k*	*1*	*c*
k-1	*1/k*	*2*	*2c*
k-2	*1/k*	*3*	*3c*
...	...	...	..
3	*1/k*	*k-2*	*(k-2)c*
2	*1/k*	*k-1*	*(k-1)c*
1	*1/k*	*k*	*kc*

Insertion Sort (Average Case Scenario)

Let $T_e(k)$ be the ***expected cost of inserting key into subrray of k elements***. Then

$$T_e(k) = (1/k)c + (1/k).2c + +(1/k).3c + \ldots\ldots\ldots + (1/k).(k-1)c + (1/k).kc$$
$$= (c/k)[1+2+3+\ldots\ldots\ldots\ldots + (k-1) + k] = (c/k)[k(k+1)/2]$$
$$= c(k+1)/2$$

The ***total cost of insertions into the complete array of size n*** is determined by summing expected costs for subarrays. If $T_a(n)$ is the average running time for the insertion sort then

$$T_a(n) = c.[\ \sum_{k=2}^{n}(k+1)/2 = \sum_{k=2}^{n} k/2 + \sum_{k=2}^{n} 1/2\ ]$$

By evaluating the summations,

$$T_a(n) = c.(n^2 + 3n - 4)/4 = \theta(n^2)$$

The ***expected running time*** of insertion sort is $\boldsymbol{\theta(n^2)}$

Selection Sort

Selection Sort

- The Idea
  - Find the smallest element in the array
  - Exchange it with the element in the first position
  - Find the second smallest element and exchange it with the element in the second position
  - Continue until the array is sorted
- Disadvantage:
  - Running time depends only slightly on the amount of order in the file

Selection Sort Illustration

- Find the smallest element in the unsorted portion of array

6	4	2	9	3

  - Interchange the smallest element with the one at the front of the unsorted portion

2	4	6	9	3

- Find the smallest element in the unsorted portion of array

2	4	6	9	3

  - Interchange the smallest element with the one at the front of the unsorted portion

2	3	6	9	4

Selection Sort Illustration (Cont...)

- Find the smallest element in the unsorted portion of array

2	3	6	9	4

  - Interchange the smallest element with the one at the front of the unsorted portion

2	3	4	9	6

- Repeat the above step until the input is sorted

2	3	4	9	6

2	3	4	6	9

- n - 1 repetitions required
  - Last element is automatically sorted

Selection Sort Algorithm

```
SELECTION-SORT(A)
1.  n ← length[A]
2.  for j ← 1 to n - 1
3.      do smallest ← j
4.       for i ← j + 1 to n
5.               do if A[i] < A[smallest]
6.               then smallest ← i
7.       exchange A[j] ↔ A[smallest]
```

Cost	Times
C_1	1
C_2	n
C_3	n – 1
C_4	$\sum_{j=1}^{n-1}(n-j+1)$
C_5	$\sum_{j=1}^{n-1}(n-j)$
C_6	$\sum_{j=1}^{n-1}(n-j)$
C_7	n – 1

$$T(n)=c_1+c_2n+c_3(n-1)+c_4\sum_{j=1}^{n-1}(n-j+1)+c_5\sum_{j=1}^{n-1}\left(n-j\right)+c_6\sum_{j=2}^{n-1}\left(n-j\right)+c_7(n-1)=\Theta(n^2)$$

Selection Sort Algorithm

SelectionSort (A[1..n])	cost	times
1 for i <- 1 to n-1	c_1	n
2 min <- i	c_2	$n-1$
3 for j <- i+1 to n	c_3	$\sum_{i=1}^{n-1}(i+1)$
4 if A[j] < A[min]	c_4	$\sum_{i=1}^{n-1} i$
5 min <- j	c_5	$\sum_{i=1}^{n-1} i\, t_{i,j}$
6 swap A[i] <--> A[min]	c_6	$n-1$

- Thus we have that $T(n) = \Theta(n^2)$

Homework

- Analyze the following Algorithms
  - Bubble Sort
  - External Sort
  - Shell Sort
  - Etc…