Sorting

CS 202 – Fundamental Structures of Computer Science II
Bilkent University
Computer Engineering Department

Sorting

- Sorting is ordering a set of elements in increasing or decreasing order.
- We will assume that
  - Elements are comparable
  - They are kept in an array
  - Each cell of the array keep one element
  - For simplicity the elements are integers. But the same methods are valid for any type of element that can be ordered.
  - We will express the number of element to be sorted as N.

Sorting

- There are various sorting algorithms
  - Easy algorithms: $O(N^2)$ running time
    - Insertion sort, etc.
  - Very easy to implement ones: $o(N^2)$
    - Efficient in practice
  - More complicated ones
    - Running time of O(NlogN)
    - Such as Quick Sort, Merge Sort, etc.
- A general purpose sorting algorithm requires Ω(NlogN) comparisons.

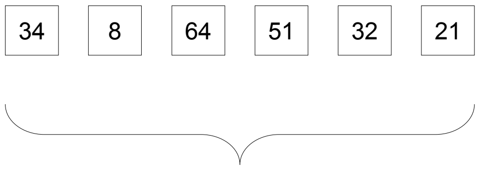
Sorting

- The data to be sorted can fit in memory;
  - We will first see the algorithms for this case.
- The data can also be residing in disk and algorithm can be run over disk
  - This is called external sorting.

Insertion Sort

- A simple algorithm
- Requires N-1 passes over the array to be sorted (of size N).
- For passes p=1 to N
  - Ensures that the elements in positions 0 through p are in sorted order.

Example

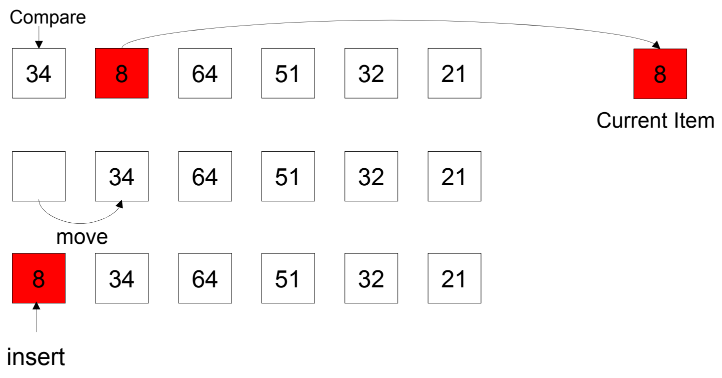


34	8	64	51	32	21

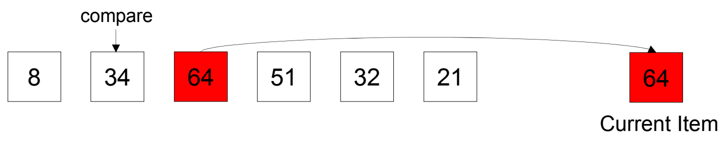
Array to be sorted.

N = 6

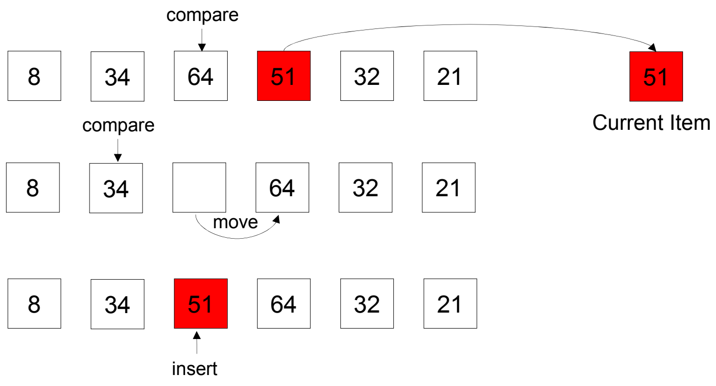
Pass 1


Compare
34
8
64
51
32
21
8
Current Item
34
64
51
32
21
move
8
34
64
51
32
21
insert

Pass 2


compare
8
34
64
51
32
21
64
Current Item

Pass 3


compare

8 34 64 51 32 21

51

Current Item

compare

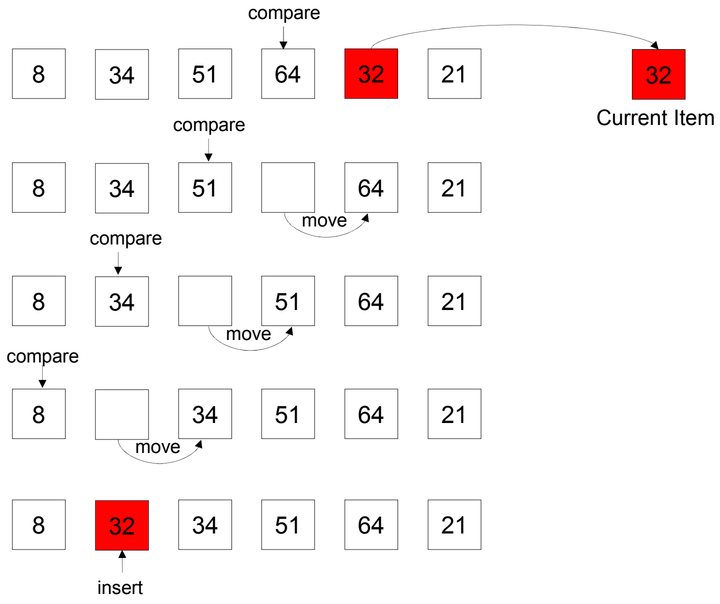
8 34   64 32 21

move

8 34 51 64 32 21

insert

Pass 4


compare

8 34 51 64 32 21

32

Current Item

compare

8 34 51   64 21

move

compare

8 34   51 64 21

move

compare

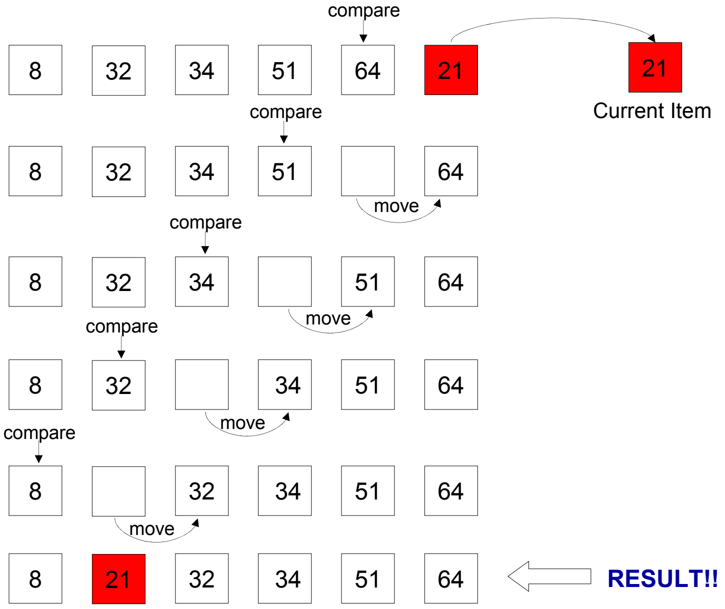
8   34 51 64 21

move

8 32 34 51 64 21

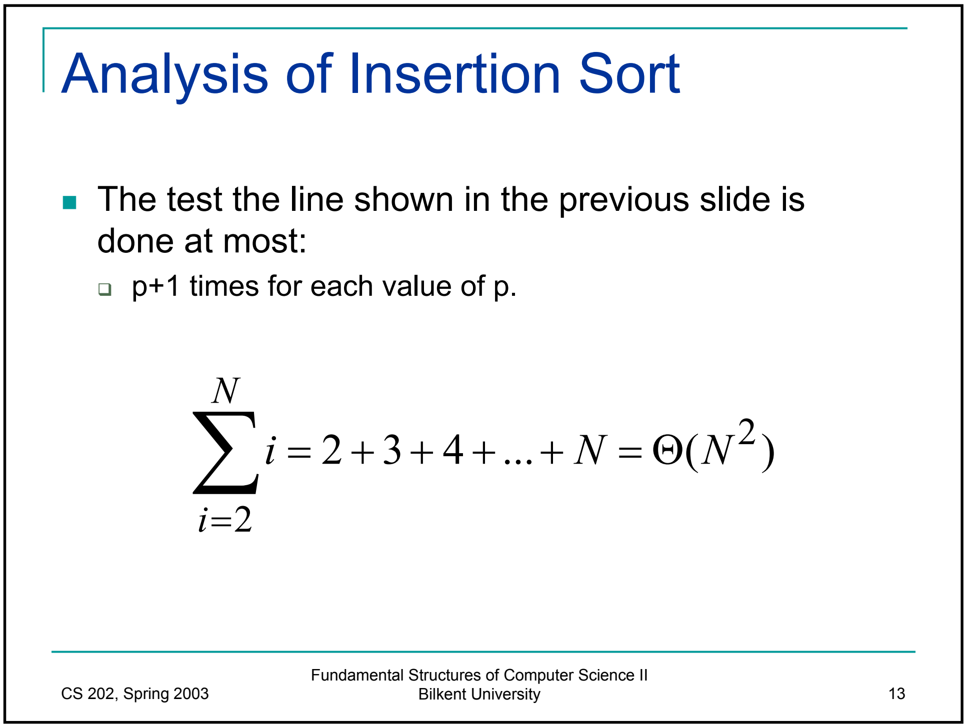
insert

Pass 5



Pseudo-Code

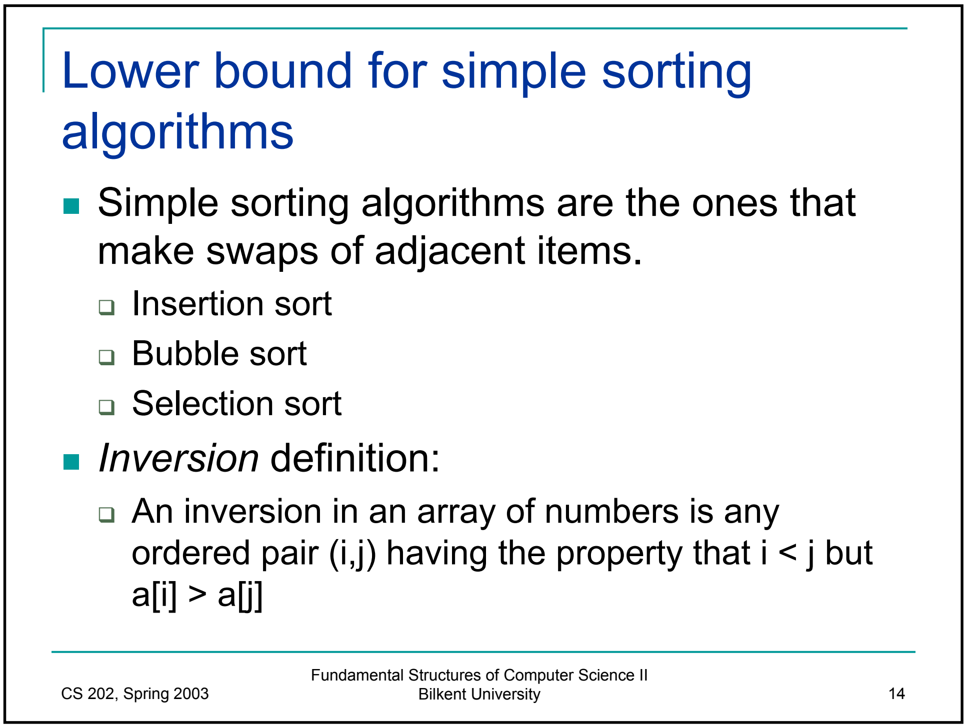
```
void
insertionSort(vector<int> &a)
{
        int j;

        for ( int p = 1; p < a.size(); ++p )
        {
                int tmp = a[p];
test  ------>   for (j=p; j > 0 && tmp < a[j-1]; j--)  /* compare */
                        a[j] = a[j-1]; /*move */
                a[j] = tmp; /* insert */
        }
}
```



Analysis of Insertion Sort

- The test the line shown in the previous slide is done at most:
  - p+1 times for each value of p.

$$\sum_{i=2}^{N} i = 2+3+4+...+N = \Theta(N^2)$$



Lower bound for simple sorting algorithms

- Simple sorting algorithms are the ones that make swaps of adjacent items.
  - Insertion sort
  - Bubble sort
  - Selection sort
- *Inversion* definition:
  - An inversion in an array of numbers is any ordered pair (i,j) having the property that i < j but a[i] > a[j]

Inversion

- Example:
  - Array items: 34 8 64 51 32 21
  - Inversions:
    - (34,8), (34,32), (34,21), (64,51), (64,32), (64,21), (51,32), (51,21), and (32,21).
    - We have total of 9 inversions.
  - *Each inversion requires a swap* in insertion sort to order the list.
  - A sorted array has no inversions.
  - Running time = O(I + N), where I is number of inversions.

Inversion

- Compute the average number of inversions in an array.
  - Assume no duplicates in the array (or list).
  - Assume there are N elements in range [1,N].
- Then input to the sorting algorithms is a permutation of these N distinct elements.

Theorem

- **Theorem**: *The average number of inversions in an array of N distinct elements is N(N-1)/4.*
- **Proof***:*
  - For any list of items, L, consider the list in reverse order $L_{r.}$
    - $L\ = 34\ \boxed{8\ \ 64}\ 51\ \ 32\ \ 21$
    - $L_r = 21\ \ 32\ \ 51\ \boxed{64\ \ 8}\ 34$
  - Consider any pair (x,y) in list L, with x < y.
  - The pair (x,y) is certainly an inversion in one of the lists L and L_r

---

Theorem

- Proof continued
  - The total number of these pairs (which are inversions) in a list L and its reverse L_r is N(N-1)/2.
  - Therefore, an average list L has half of this amount, which is N(N-1)/4.

Shell Sort

- Invented by Donald Shell.
- Also referred to as *diminishing increment sort*.
- Shell sort uses a sequence h_1, h_2, …, h_t, called the increment sequence.
  - h_1 must be 1.
  - Any sequence will do.

Shell Sort

- It is executed in phase.
  - One phase for each h_k
- After a phase where increment were h_k
  - For every i, a[i] <= a[i+h_k].
  - This means all elements spaced h_k apart are sorted.
  - The input is then said to be *h_k sorted*.
- An h_k sorted input, which is then h_{k-1} sorted, is still h_k sorted.

Shell Sort

Original List	81	94	11	96	12	35	17	95	28	58	41	75	15
After 5-sort	37	17	11	28	12	41	75	15	96	58	81	94	95
After 3-sort	28	12	11	35	15	41	58	17	94	75	81	96	95
After 1-sort	11	12	15	17	28	35	41	58	75	81	94	95	96

Shellsort Algorithm

```
void shellsort (vector<int> &a)
{
        int j, i;
        int gap;

        for (gap = a.size() /2; gap > 0; gap /=2)
        {
                for (i=gap; i < a.size(); i++)
                {
                        int tmp = a[i];
                        for (j=i; j>=gap && tmp < a[j-gap]; j -= gap)
                                a[j] = a[j-gap];
                        a[j] = tmp;
                }
        }
}
```

Choosing Increment Sequence

$$h_t = \left\lfloor \frac{N}{2} \right\rfloor$$

$$h_k = \left\lfloor \frac{h_{k+1}}{2} \right\rfloor$$

$$h_1 = 1$$

Suggested by Donald Shell
N: the number of items to sort.

Worst Case Analysis of Shell Sort

- Theorem:
  - *The worst case running time of Shell sort using Shell's increments is $\Theta(N^2)$.*
- .
  - We will show a *lower bound* for the running time.
  - We will also show an *upper bound* for the running time.

Lower bound

- We will show that there exists an input that causes the algorithm to run in $\Omega(N^2)$ time.
  - Assume N is power of 2.
  - Assume these N elements is stored in an array indexed from 1 to N.
  - Assume that
    - odd index values contain the N/2 largest elements and
    - even index values contain the N/2 smallest element.
- 1,9,2,10,3,11,4,12,5,13,6,14,7,15,8,16
  is such as sequence.

Lower bound

- Shell's increments are:
  - 1, 2, 3, …., N/2
- All increments except the last one are even.
- When we come to the last pass,
  - all largest items are in even positions and
  - all smallest items are in odd positions.
- Snapshot before last pass
  - 1,9,2,10,3,11,4,12,5,13,6,14,7,15,8,16

Lower bound

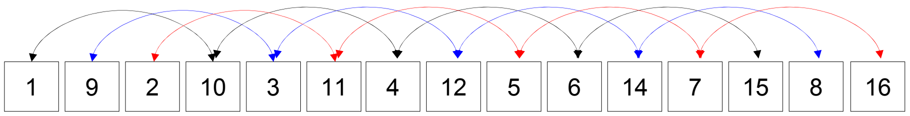
- The i^{th} smallest number is at position 2i-1 before the last pass.
- Restoring the ith element to its correct position requires:
    - 2i-1-i = i-1 moves towards the beginning of the array (each move make the item go one cell left).
- Therefore to place N/2 smallest elements to their correct positions require amount of work in the order:

$$\sum_{i=1}^{N/2} i-1=\Omega(N^2)$$

Upper Bound

- A pass with increment h_k consists of h_k insertion sorts of about N/h_k elements

$h_k = 3$,   N = 16



━━ Insertion sort of 16/3 ~= 5 items, items are = 1, 10, 4, 6, 15

━━ Insertion sort of 16/3 ~= 5 items, items are = 9, 3, 12, 14, 8

━━ Insertion sort of 16/3 ~= 5 items

$$\boxed{h_k * (N/h_k)^2}$$

Upper Bound

- Summing over all passed

$$\sum_{i=1}^{t} N^2 / h_i = O(N^2 \sum_{i=1}^{t} 1 / h_i) = \boxed{O(N^2)}$$

since $\sum_{i=1}^{t} 1 / h_i < 2$