Course Name: CS302-Design and analysis of Algorithm

Credit Hours: 3

Week-2

Classes of Complexities

- Constant: $O(c)$,
- Logarithmic: $O(\log_c n)$,
- Linear: $O(n)$,
- Quadratic: $O(n^2)$,
- Cubic: $O(n^3)$,
- Polynomial: $O(n^c)$
- Exponential: $O(c^n)$

- Sorting techniques and their analysis (I):
  - Insertion sort
  - Bubble sort
  - Selection sort

Introduction

- The sorting problem is to arrange a sequence of records so that the values of their key fields form a non-decreasing sequence.

- Given records $r_1, r_1, \ldots. r_n$ with key values $k_1, k_1, \ldots. k_n$, respectively we must produce the same records in an order $r_{i1}, r_{i2}, \ldots. r_{in}$ such that the keys are in the corresponding non-decreasing order.

  $$key(r_{i1}) \leq key(r_{i2}) \leq key(r_{i3}) \leq key(r_{i4}) \leq key(r_{i5}) \leq key(r_{i6})$$

  $$k_{i1} \leq k_{i2} \leq k_{i3} \leq k_{i4} \leq k_{i5} \leq k_{i6}$$

- The records may NOT have distinct values, and can appear in any order.

- Different criteria to evaluate the running time, as follows:
  - Number of algorithm steps.
  - Number of comparisons between the keys (for expensive comparisons).
  - The number of times a record is moved (for large records).

Bubble Sort

- One of the simplest sorting methods.
- The basic idea is the “weight” of the record.
- The records are kept in an array.
- “heavy” records bubbling up to the end.
- We make repeated passes over the array and sort the values
- If two adjacent elements are out of order, we reverse the order.

Bubble Sort

- The overall effect, is that after the first pass the “heavy” record will bubble all the way to the end.
- On the second top pass, the second highest value goes to the second position, and so on.
- Reduce 1 element in each iteration

Bubble Sort

```
int n =N; // N is the
size of the array;

 for (int i = 0; i < N; i++){  for
(int j = 1; j < n; j++) {  if (A[j]
    < A[j-1]) {  swap(j-1 , j);
     }//end if
```

```
} //end inner for
} //end inner for
```

Complexity ?

$O(N^2)$

Algorithm does not exit until all the data is checked

```
swap ( x , y) {
 int temp = A[x];
A[x] = A[y];
A[y] = temp;
}
```

// Swap function assumes that // A[n] is a globally declared array

Bubble Sort

```
for (int j = 1; j < n; j++) {  if
    (A[j] < A[j-1]) {
  swap(j-1 , j);
```

```
int n =N; // N is the size of the array;
for (int i = 0; i < N; i++){
 int swapped = 0;
```

// Swap function assumes that // A[n] is a globally declared array

```
swap ( x , y) {
```

```
 int temp = A[x];
A[x] = A[y];
A[y] = temp;
}

 swapped = 1;
 }//end if
 } //end inner for
 n = n-1;
 if (swapped == 0)
 break;
 } //end inner for
```

Complexity ?

$O(N^2)$

No bubbling to the top position, because the lightest record is already there.

Algorithm exits if no swap done in previous (outer loop) step

Bubble Sort Example (First Pass)

62 (0) 58 (1) 55 (2) 10 (3) 45 (4) 44 (5) 6 (6) 90 (7) *i* = 0

j = 1 58 (0) 62 (1) 55 (2) 10 (3) 45 (4) 44 (5) 6 (6) 90 (7) 58 (0) 55 (1) 62 (2) 10 (3) 45 (4) 44 (5)

6 (6) 90 (7) *j* = 2 58 (0) 55 (1) ~~10 (2) 62 (3)~~ 45 (4) 44 (5) 6 (6) 90 (7) *j* = 3 58 (0) 55 (1) 10 (2)

45 (3) 62 (4) 44 (5) 6 (6) 90 (7) *j* = 4 58 (0) 55 (1) 10 (2) 45 (3) 44 (4) 62 (5) 6 (6) 90 (7) *j* = 5

58 (0) 55 (1) 10 (2) 45 (3) 44 (4) 6 (5) 62 (6) ~~90~~ (7) *j* = 6

58 (0) 55 (1) 10 (2) 45 (3) 44 (4) 6 (5) 62 (6) 90 (7) *j* = 7

```
int n =N; // N is the size of the array;
 for (int i = 0; i < N; i++){
 int swapped = 0;
 for (int j = 1; j < n; j++)
 if (A[j] < A[j-1]) {
 swap(j-1 , j); swapped = 1;
 }//end if
 n = n-1;
 if (swapped == 0)
 break; }//End outer for
```

swap(*A*[0] , *A*[1]) swap(*A*[1] , *A*[2]) swap(*A*[2] , *A*[3]) swap(*A*[3] , *A*[4]) swap(*A*[4] , *A*[5])

swap(*A*[5] , *A*[6]) 9

Bubble Sort Example (Second Pass)

i = 1

j = 1

j = 2

j = 3

6 (4) 58 (5) 62 (6) 90 (7)

j = 4

j = 5

j = 6

58 (0) 55 (1) 10 (2) 45 (3) 44 (4) 6 (5)
62 (6) 90 (7)

---

55 (0) 58 (1) 10 (2) 45 (3) 44 (4) 6 (5)

62 (6) 90 (7)

55 (0) 10 (1) ~~58~~ (2) 45 (3) 44 (4) 6 (5)

62 (6) 90 (7) 55 (0) 10 (1) ~~45~~ (2) ~~58~~ (3)

44 (4) 6 (5) 62 (6) 90 (7) 55 (0) 10 (1)

45 (2) 44 (3) 58 (4) ~~6~~ (5) 62 (6) 90 (7)

55 (0) 10 (1) 45 (2) 44 (3) 6 (4) 58 (5)

62 (6) 90 (7) 55 (0) 10 (1) 45 (2) 44 (3)

```
int n =N; // N is the size of the array;
 for (int i = 0; i < N; i++){
 int swapped = 0;
 for (int j = 1; j < n; j++)
 if (A[j] < A[j-1]) {
 swap(j-1 , j); swapped = 1;
 }//end if
 n = n-1;
 if (swapped == 0)
 break; }//End outer for
```

swap(*A*[0] , *A*[1]) swap(*A*[1]

, *A*[2])

swap(*A*[2] , *A*[3]) swap(*A*[3]

, *A*[4]) swap(*A*[4] , *A*[5])

Bubble Sort Example (Third Pass)

***i* = 2**

55 (0) 10 (1) 45 (2) 44 (3) 6 (4) 58 (5) 62 (6) 90 (7)

***j* = 1**

***j* = 2**

j = 3

j = 4

j = 5

10 (0) 45 (1) 44 (2) 6 (3) 55 (4) 58 (5) 62 (6) 90 (7)

```
int n =N; // N is the size of the array;
 for (int i = 0; i < N; i++){
 int swapped = 0;
 for (int j = 1; j < n; j++)
 if (A[j] < A[j-1]) {
 swap(j-1 , j); swapped = 1;
 }//end if
 n = n-1;
 if (swapped == 0)
 break; }//End outer for
```


Bubble Sort Example (Fourth Pass)

i = 3

10 (0) 45 (1) 44 (2) 6 (3) 55 (4) 58 (5) 62 (6) 90 (7)

j = 1

j = 2

j = 3

j = 4

10 (0) 44 (1) 6 (2) 45 (3) 55 (4) 58 (5) 62 (6) 90 (7)

```
int n =N; // N is the size of the array;
 for (int i = 0; i < N; i++){
 int swapped = 0;
 for (int j = 1; j < n; j++)
 if (A[j] < A[j-1]) {
 swap(j-1 , j); swapped = 1;
 }//end if
 n = n-1;
 if (swapped == 0)
 break; }//End outer for
```

Bubble Sort Example (Fifth Pass)

i = 4

6 (1) 44 (2) 45 (3) 55 (4) 58 (5) 62 (6) 90 (7)

j = 1

j = 2

j = 3

```
int n =N; // N is the size of the array;
 for (int i = 0; i < N; i++){
 int swapped = 0;
 for (int j = 1; j < n; j++)
 if (A[j] < A[j-1]) {
 swap(j-1 , j); swapped = 1;
 }//end if
 n = n-1;
 if (swapped == 0)
 break; }//End outer for
```

10 (0) 44 (1) 6 (2) 45 (3) 55 (4) 58 (5) 62 (6) 90 (7) 10 (0)

Bubble Sort Example (Sixth Pass)

i = 5

j = 1

j = 2

```
if (A[j] < A[j-1]) {
swap(j-1 , j); swapped = 1;
}//end if
n = n-1;
if (swapped == 0)
break; }//End outer for
```

10 (0) 6 (1) 44 (2) 45 (3) 55 (4) 58 (5) 62 (6) 90 (7) 6 (0) 10

(1) 44 (2) 45 (3) 55 (4) 58 (5) 62 (6) 90 (7)

```
int n =N; // N is the size of the array;
 for (int i = 0; i < N; i++){
 int swapped = 0;
 for (int j = 1; j < n; j++)
```

Bubble Sort Example (Seventh Pass)

i = 6

```
if (swapped == 0)
break; }//End outer for
```

j = 1

6 (0) 10 (1) 44 (2) 45 (3) 55 (4) 58 (5) 62 (6) 90 (7) 6(0) 10

(1) 44 (2) 45 (3) 55 (4) 58 (5) 62 (6) 90 (7)

```
int n =N; // N is the size of the array;
 for (int i = 0; i < N; i++){
 int swapped = 0;
 for (int j = 1; j < n; j++)
 if (A[j] < A[j-1]) {
 swap(j-1 , j); swapped = 1;
 }//end if
 n = n-1;
```

Bubble Sort

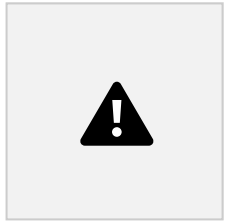
```
int n =N; // N is the size of the array;
 for (int i = 0; i < N; i++){
      for (int j = 1; j < n; j++) {
     if (A[j] < A[j-1]) {
     swap(j-1 , j);
      }//end if
 } //end inner for
} //end inner for
```

Complexity ?

$O(N^2)$

Algorithm does not exit until all the data is checked

Bubble Sort



```
int n =N; // N is the size of the array;
 for (int i = 0; i < N; i++){
 int swapped = 0;
 for (int j = 1; j < n; j++) {
```

```
      if (A[j] < A[j-1]) {
      swap(j-1 , j);
      swapped = 1;
      }//end if
 } //end inner for
 n = n-1;
 if (swapped == 0)
 break;
 } //end inner for
```

Complexity ?

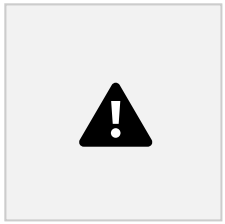
$O(N^2)$

Algorithm exits if no swap done in previous (outer loop) step

Insertion Sort

- On the *i*th pass we "insert" the *i*th element *A*[i] into its rightful place among *A*[1],*A*[2],…*A*[i-1] which were placed in sorted order.
- After this insertion *A*[1],*A*[2],…*A*[i] are in sorted order.
-

Insertion Sort



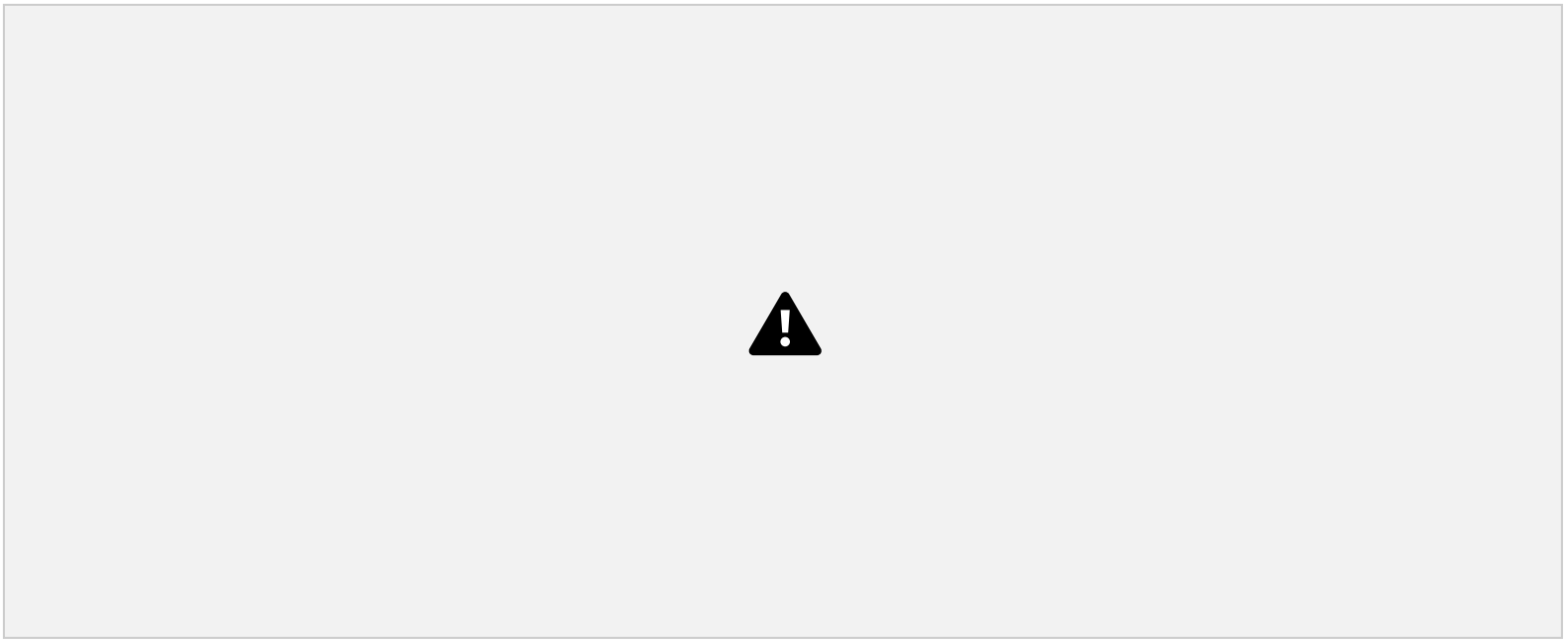
```
for (int i =1, i < n, i++){
 temp = A[i];
     for (int j = i, j>0 && A[j-1] > temp, j--)
      A[j] = A[j-1]
 A[j] = temp;
```

}// end outer for

Complexity ?

$O(N^2)$

Insertion Sort



Insertion Sort Example (First Pass)

$i = 1$

62 (0) 58 (1) 55 (2) 10 (3) 45 (4) 44 (5) 6 (6) 90 (7)

temp = 58

j = 1 62 (0) 62 (1) 55 (2) 10 (3) 45 (4) 44 (5) 6 (6)

90 (7) 62 > 58

j = 0 58 (0) 62 (1) 55 (2) 10 (3) 45 (4) 44 (5) 6 (6)

90 (7) ◆◆ *j* > 0 && A [*j* -1] > *temp*

A[1] = *A*[0]

◆◆ *A*[1] = *A*[0]

◆◆ *j* = 0 ◆◆ exit

A[*j*] = temp (= 58)

```
for (int i =1, i < n, i++){
 temp = A[i];
for (int j = i, j > 0 && A[j-1] > temp,
j--)  A[j] = A[j-1]
 A[j] = temp;
```

A[0] = temp = 58;

}// end outer for

Insertion Sort Example (Second Pass)

58 (0) 62 (1) 55 (2) 10 (3) 45 (4) 44 (5) 6 (6) 90 (7) *i* = 2

58 (0)

62 > 55

j = 2 58 (0) 62 (1) 62 (2) 10 (3) 45 (4) 44 (5) 6 (6) 90 (7)

58 > 55

j = 1 58 (1) 62 (2) 10 (3) 45 (4) 44 (5) 6 (6) 90 (7) *j* = 0

55 (0) 58 (1) 62 (2) 10 (3) 45 (4) 44 (5) 6 (6) 90 (7)

temp = 55 *A*[2] = *A*[1] *A*[1] = *A*[0]

A[*0*] = *temp*
= 55;

```
    for (int i =1, i < n, i++){
temp = A[i];
      for (int j = i, j > 0 && A[j-1] > temp, j--)
       A[j] = A[j-1]
A[j] = temp;
}// end outer for
```


Insertion Sort Example (Third Pass)

55 (0) 58 (1) 62 (2) 10 (3) 45 (4) 44 (5) 6 (6) 90 (7) *i* = 3

temp = 10

j = 3 55 (0) 58 (1) 62 (2) 62 (3) 45 (4) 44 (5) 6 (6) 90 (7)

55 (0)

= 1 55 (0) 55 (1) 58 (2) 62 (3) 45 (4) 44 (5) 6 (6) 90 (7) *j* = 0 10 (0) 55 (1) 58 (2) 62 (3) 45 (4) 44 (5) 6 (6) 90 (7)

A[3] = *A*[2] *A*[2] = *A*[1] *A*[1] = *A*[0]

j = 2 58 (1) 58 (2) 62 (3) 45 (4) 44 (5) 6 (6) 90 (7) *j*

A[*0*] = *temp* = 55;

```
for (int i =1, i < n, i++){
temp = A[i];
    for (int j = i, j > 0 && A[j-1] > temp, j--)
     A[j] = A[j-1]
A[j] = temp;
}// end outer for
```

Insertion Sort Example (Fourth Pass)

10 (0) 55 (1) 58 (2) 62 (3) 45 (4) 44 (5) 6 (6) 90 (7) *i* = 4 *temp* = 45

j = 4 10 (0) 55 (1) 58 (2) 62 (3) 62 (4) 44 (5) 6 (6) 90 (7)

10 (0)

j = 3 55 (1) 58 (2) 58 (3) 62 (4) 44 (5) 6 (6) 90 (7)

j = 2 10 (0) 55 (1) 55 (2) 58 (3) 62 (4) 44 (5) 6 (6) 90 (7) 10 > 45

j = 1 10 (0) 45 (1) 55 (2) 58 (3) 62 (4) 44 (5) 6 (6) 90 (7)

A[4] = *A*[3] *A*[3] = *A*[2] *A*[2] = *A*[1]

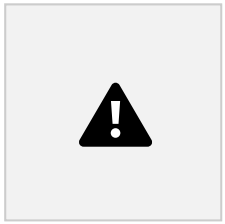
A[1] = *temp* = 45;

```
for (int i =1, i < n, i++){
temp = A[i];
    for (int j = i, j > 0 && A[j-1] > temp , j--)
```

$A[j] = A[j\text{-}1]$

$A[j] = \text{temp};$

}// end outer for



Insertion Sort Example (Fifth Pass)

10 (0) 45 (1) 55 (2) 58 (3) 62 (4) 44 (5) 6 (6) 90 (7) *i* = 5 *temp* = 44

j = 5 10 (0) 45 (1) 55 (2) 58 (3) 62 (4) 62 (5) 6 (6) 90 (7)

10 (0)

= 3 10 (0) 45 (1) 55 (2) 55 (3) 58 (4) 62 (5) 6 (6) 90 (7)

A[5] = *A*[4] *A*[4] = *A*[3] *A*[3] = *A*[2]

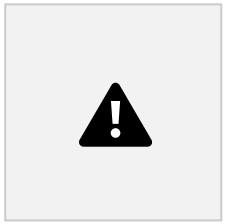
j = 4 45 (1) 55 (2) 58 (3) 58 (4) 62 (5) 6 (6) 90 (7) *j*

j = 2 10 (0) 45 (1) 45 (2) 55 (3) 58 (4) 62 (5) 6 (6) 90 (7) *A*[2] = *A*[1]

j = 1 10 (0) 44 (1) 45 (2) 55 (3) 58 (4) 62 (5) 6 (6) 90 (7) *A*[1] = *temp*

= 44;

```
    for (int i =1, i < n, i++){
temp = A[i];
      for (int j = i, j > 0 && A[j-1] > temp, j--)
       A[j] = A[j-1]
A[j] = temp;
}// end outer for
```

Insertion Sort Example (Sixth Pass)

10 (0) 44 (1) 45 (2) 55 (3) 58 (4) 62 (5) 6(6) 90 (7) *i* = 6 *temp* = 6

j = 6 10 (0) 44 (1) 45 (2) 55 (3) 58 (4) 62 (5) 62(6) 90 (7)

10 (0)

j = 5 44 (1) 45 (2) 55 (3) 58 (4) 58 (5) 62(6) 90 (7)

j = 4 10 (0) 44 (1) 45 (2) 55 (3) 55 (4) 58 (5) 62(6) 90 (7)

~~*A*~~[6] = *A*[5] *A*[5] = *A*[4] *A*[4] = *A*[3]

j = 3 10 (0) 44 (1) 45 (2) 45 (3) 55 (4) 58 (5) 62 (6) 90 (7) ***A*[3] = *A*[2]**

j = 2 10 (0) 44 (1) 44 (2) 45 (3) 55 (4) 58 (5) 62 (6) 90 (7) *j* = 1 10 (0) 10 (1) 44 (2) 45 (3) 55 (4) 58 (5) 62 (6) 90 (7) *j* = 0 6 (0) 10 (1) 44 (2) 45 (3) 55 (4) 58 (5) 62 (6) 90 (7)

***A*[2] = *A*[1] *A*[1] = *A*[0]**

```
temp = A[i];
for (int j = i, j > 0 && A[j-1] > temp,
j--)  A[j] = A[j-1]
A[j] = temp;
```

***A*[0] = *temp* = 6;**

```
for (int i =1, i < n, i++){
```



```
}// end outer for
```

Insertion Sort Example (Sixth Pass)

6 (0) 10 (1) 45 (2) 45 (3) 55 (4) 58 (5) 62 (6) 90 (7) ***i* = 7** ***temp* = 90**

A[j-1] > temp?
No
Quit

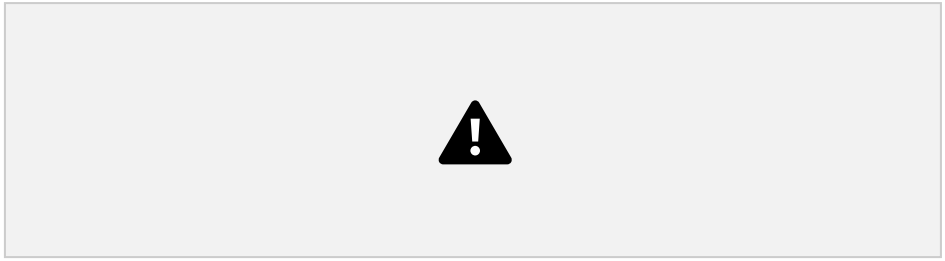
```
for (int i =1, i < n, i++){
 temp = A[i];
for (int j = i, j > 0 && A[j-1] > temp, j--)
 A[j] = A[j-1]
 A[j] = temp;
 }// end outer for
```

Analysis of Insertion Sort

Because of the nested loops, each of which can take n iterations, insertion sort is $O(n^2)$.

- Furthermore, this bound is tight, because input in reverse order can actually achieve this bound.

- A precise calculation shows that the test at line 3 can be executed at most i times for each value of i. Summing over all i gives a total of



- If the input is presorted, the running time is $O(n)$.
  because the test in the inner for loop always fails

immediately

- The average running time also $O(n^2)$

Selection Sort

- Find the minimum value in the list
- Swap it with the value in the first position
- Repeat the steps above for the remainder of the list (starting at the second position and advancing each time)

http://en.wikipedia.org/wiki/Selection_sort

Selection Sort

```
for (int i =0, i < n, i++){
 min = i;
    for (int j = i+1, j < n ,
```

```
j++){
    if ( A[j] < A[min]){
    min = j
 } // end if  }// end
 inner for  swap(i,
min)   }// end outer
           for
```

Complexity ?

$O(N^2)$

```
// Swap function assumes that //
A[n] is a globally declared array
swap(i, min) {
 int temp = A[i];
A[i] = A[min];
A[min] = temp;
}
```


~~Selection Sort - Example~~

```
for (int j = i+1, j < n , j++){
if ( A[j] < A[min]){
min = j
} // end if
}// end inner for
swap(i, min)
}// end outer for
```

7 2 7 8 5 4 2 4 8 5 7 2 4

5 8 7 2 4 5 7 8

- The Selection Sort might swap an array element with itself--this is harmless.

```
for (int i =0, i < n, i++){
 min = i;
```

[redacted] Sort Example

i = 0 **min = 6**

62 (0) 58 (1) 55 (2) 10 (3) 45 (4) 44 (5) 6 (6) 90 (7)

A[min] = 6

A[min] = 62 A[min] = 58A[min] = 55A[min] = 10 A[min] = 10A[min] = 10A[min] = 6A[min] = 6 6

(0) 58 (1) 55 (2) 10 (3) 45 (4) 44 (5) 62 (6) 90 (7) $i = 1$ min = 3 A[min] = 10

A[min] = 58A[min] = 55A[min] = 10 A[min] = 10A[min] = 10A[min] = 10A[min] = 10

6 (0) 10 (1) 55 (2) 58 (3) 45 (4) 44 (5) 62 (6) 90 (7) $i = 2$ min = 5 A[min] = 44

A[min] = 55A[min] = 58 A[min] = 45A[min] = 44A[min] = 44A[min] = 44

6 (0) 10 (1) 44 (2) 58 (3) 45 (4) 55 (5) 62 (6) 90 (7) $i = 3$ min = 4 A[min] = 45

A[min] = 58 A[min] = 45A[min] = 45A[min] = 45A[min] = 45

```
for (int i =0, i < n, i++){
 min = i;
       for (int j = i+1, j < n , j++){
             if ( A[j] < A[min]){
             min = j
 } // end if
 }// end inner for
       swap(i, min)
 }// end outer for
```


[redacted] Sort Example - continued i =

62 (0)	58 (1)

0 min = 6

6 (0)	58 (1)
6 (0)	10 (1)

55 (2) 10 (3) 45 (4) 44 (5) 6 (6) 90 (7)

A[min] = 6

55 (2)

10 (3)

45 (4) 44 (5) 62 (6) 90 (7) i = 1 min = 3

A[min] = 10

55 (2) 58 (3) 45 (4) 44 (5) 62 (6) 90 (7) i = 2 min = 5

A[min] = 44

6 (0) 10 (1) 44 (2) 58 (3) 45 (4) 55 (5) 62 (6) 90 (7) i = 3 min = 4 A[min] = 45

6 (0) 10 (1) 44 (2) 45 (3) 58 (4) 55 (5) 62 (6) 90 (7) i = 4 min = 55 A[min] = 45

A[min] = 58A[min]= 55A[min] = 55A[min] = 55

6 (0) 10 (1) 44 (2) 45 (3) 55 (4) 58 (5) 62 (6) 90 (7) i = 5 min = 5 A[min] = 58

6 (0) 10 (1) 44 (2) 45 (3) 55 (4) 58 (5) 62 (6) 90 (7) i = 6 min = 6 A[min] = 62

6 (0) 10 (1) 44 (2) 45 (3) 55 (4) 58 (5) 62 (6) 90 (7) i = 7 min = 6 A[min] = 62

```
for (int i =0, i < n, i++){
 min = i;
        for (int j = i+1, j < n , j++){
              if ( A[j] < A[min]){
```

```
            min = j
} // end if
}// end inner for
        swap(i, min)
}// end outer for
```


Selection Sort vs Insertion Sort

- Selection sort's advantage is that
  - While insertion sort typically makes fewer comparisons than selection sort,
  - Insertion sort requires more writes than the selection sort because the inner loop of the insertion sort can require shifting large sections of the sorted portion of the array.
    - In general, insertion sort will write to the array $O(n^2)$ times
    - Whereas selection sort will write/swap only O(n) times

- For this reason selection sort may be preferable in cases where writing to memory is significantly more expensive than reading,
  - such as with EPROM or flash memory

Comparisons of different sorting algorithms – Home Work