Preview

- Running time analysis
  - Selection Sort
  - Insertion Sort
  - Bubble Sort
  - Shell Sort

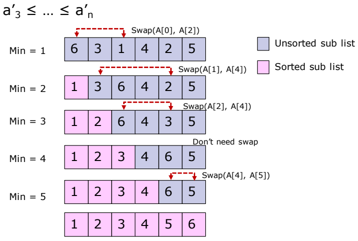
Running Time Analysis with Sorting Algorithms
(Selection Sort)

- For each iteration, a minimum is found in the sub list and it is placed into right position in sorted sub list.

```
Selection sort (A)
{
  for j = 0 |A|-1 do                      C1
   {
        Min = j                           C2
        for k= j+1 to |A|-1               C3
        {
            if A{k} <A[Min] then          C4
                  Min = k                 C5
        }
        If Min ≠ j                        C6
            Swap (A[j], A[Min])           C7
   }
}
```


Running Time Analysis with Sorting Algorithms
(Selection Sort)

- Input: A sequence of n numbers $<a_1, a_2, a_3, \ldots a_n>$
- Output: Permutation $<a'_1, a'_2, a'_3, \ldots a'_n>$ of input sequence such that $a'_1 \le a'_2 \le a'_3 \le \ldots \le a'_n$



Running Time Analysis with Sorting Algorithms
(Selection Sort)

- For each iteration, a minimum is found in the sub list and it is placed into right position in sorted sub list.

```
Selection sort (A)
{
  for j = 0 |A|-2 do                      C1 n
   {
        Min = j                           C2 (n − 1)
        for k= j+1 to |A|-1               C3 (n − 1)
        {
            if A[k] <A[Min] then          C4 n(n − 1)/2
                  Min = k
        }
        If Min ≠ j                        C6 (n − 1)
            Swap (A[j], A[Min])
   }
}
```


Running Time Analysis with Sorting Algorithms
(Selection Sort: worst case)

- For each iteration, a minimum is found in the sub list and it is placed into right position in sorted sub list.

```
Selection sort (A)
{
  for j = 0 |A|-2 do                      C1 n
   {
        Min = j                           C2 (n − 1)
        for k= j+1 to |A|-1               C3 (n − 1)
        {
            if A[k] <A[Min] then          C4 n(n − 1)/2
                  Min = k                 C5 n(n − 1)/2
        }
        If Min ≠ j                        C6 (n − 1)
            Swap (A[j], A[Min])           C6 (n − 1)
   }
}
```


Running Time Analysis with Sorting Algorithms
(Selection Sort)

Selection Sort Analysis

- Best Case:

$$T(n) = C_1 n + C_2(n-1) + C_3(n-1) + C_4\left(\frac{n(n-1)}{2}\right) + C_6(n-1)$$
$$= \frac{C_4}{2} n^2 + \left(C_1 + C_2 + C_3 - \frac{C_4}{2} + C_6\right) n - (C_2 + C_3 + C_6) = \Theta(n^2)$$

- Worst Case

$$T(n) = C_1 n + C_2(n-1) + C_3(n-1) + C_4\left(\frac{n(n-1)}{2}\right) + C_5\left(\frac{n(n-1)}{2}\right) + C_6(n-1) + C_7(n-1)$$
$$= (\frac{C_4}{2} + \frac{C_5}{2}) n^2 + \left(C_1 + C_2 + C_3 - \frac{C_4}{2} - \frac{C_5}{2} + C_6 + C_7\right) n - (C_2 + C_3 + C_6 + C_7) = \Theta(n^2)$$

Running Time Analysis with Sorting Algorithms
(Insertion Sort)

- Input: A sequence of n numbers $<a_1, a_2, a_3, \ldots a_n>$
- Output: Permutation $<a'_1, a'_2, a'_3, \ldots a'_n>$ of input sequence such that $a'_1 \le a'_2 \le a'_3 \le \ldots \le a'_n$

Insert 3	6	3	1	4	2	5
Insert 1	3	6	1	4	2	5
Insert 4	1	3	6	4	2	5
Insert 2	1	3	4	6	2	5
Insert 5	1	2	3	4	6	5
	1	2	3	4	5	6

Unsorted sub list

Sorted sub list

Running Time Analysis with Sorting Algorithms
(Insertion Sort)

- For each iteration, a element is placed into right position in sorted sub list.

Code	Cost
Insertion sort (A)	
{	
for j = 2 to length of array A do	C_1
{	
key = A[j]	C_2
i = j – 1	C_3
while i > 0 and A[i] >key do	C_4
{	
A[i +1] = A[i]	C_5
i = i –1	C_6
}	
A[i +1] = key	C_7
}	
}	

Running Time Analysis with Sorting Algorithms
(Insertion Sort: Best Case)

Insert 2	1	2	3	4	5	6
Insert 3	1	2	3	4	5	6
Insert 4	1	2	3	4	5	6
Insert 5	1	2	3	4	5	6
Insert 6	1	2	3	4	5	6
	1	2	3	4	5	6

Unsorted sub list

Sorted sub list

Running Time Analysis with Sorting Algorithms
(Insertion Sort: Best Case)

Code	Cost
Insertion sort (A)	
{	
for j = 2 to length of array A do	$C_1(n-1)$
{	
key = A[j]	$C_2(n-1)$
i = j – 1	$C_3(n-1)$
while i > 0 and A[i] >key do	$C_4(n-1)$
{	
A[i +1] = A[i]	
i = i –1	
}	
A[i +1] = key	$C_7(n-1)$
}	
}	

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4(n-1) + c_7(n-1)$$
$$= (c_1 + c_2 + c_3 + c_4 + c_7)n - (c_2 + c_3 + c_4 + c_7)$$

Running Time Analysis with Sorting Algorithms
(Insertion Sort: Worst Case)

Insert 5	6	5	4	3	2	1
Insert 4	5	6	4	3	2	1
Insert 3	4	5	6	3	2	1
Insert 2	3	4	5	6	2	1
Insert 1	2	3	4	5	6	1
	1	2	3	4	5	6

Unsorted sub list

Sorted sub list

Running Time Analysis with Sorting Algorithms
(Insertion Sort: Worst Case)

Code	Cost
Insertion sort (A)	
{	
for j = 2 to length of array A do	$C_1(n-1)$
{	
key = A[j]	$C_2(n-1)$
i = j – 1	$C_3(n-1)$
while i > 0 and A[i] >key do	$C_4(\frac{n(n+1)}{2} - 1)$
{	
A[i +1] = A[i]	$C_5(\frac{n(n-1)}{2})$
i = i –1	$C_6(\frac{n(n-1)}{2})$
}	
A[i +1] = key	$C_7(n-1)$
}	
}	

Running Time Analysis with Sorting Algorithms
(Insertion Sort)

Insertion Sort Analysis

- Best Case

$T(n) = C_1(n-1) + C_2(n-1) + C_3(n-1) + C_4(n-1) + C_7(n-1)$
$= (C_1 + C_2 + C_3 + C_4 + C_7)n - (C_1 + C_2 + C_3 + C_4 + C_7) = \Theta(n)$

- Worst Case:

$$T(n)$$
$$= C_1(n-1) + C_2(n-1) + C_3(n-1) + C_4\left(\frac{n(n+1)}{2} - 1\right) + C_5\left(\frac{n(n-1)}{2}\right) + C_6\left(\frac{n(n-1)}{2}\right) + C_7(n-1)$$
$$= \left(\frac{C_4}{2} + \frac{C_5}{2} + \frac{C_6}{2}\right)n^2 + \left(C_1 + C_2 + C_3 + \frac{C_4}{2} - \frac{C_5}{2} - \frac{C_6}{2} + C_7\right)n - (C_1 + C_2 + C_3 + C_4 + C_7) = \Theta(n^2)$$

Running Time Analysis with Sorting Algorithms
(Bubble Sort)

- Input: A sequence of n numbers $<a_1, a_2, a_3, \ldots a_n>$
- Output: Permutation $<a'_1, a'_2, a'_3, \ldots a'_n>$ of input sequence such that $a'_1 \le a'_2 \le a'_3 \le \ldots \le a'_n$

Bubble up	6	3	1	4	2	5
Bubble up	3	1	4	2	5	6
Bubble up	1	3	2	4	5	6
Bubble up	1	2	3	4	5	6
Bubble up	1	2	3	4	5	6
	1	2	3	4	5	6

Unsorted sub list
Sorted sub list

Running Time Analysis with Sorting Algorithms
(Bubble Sort)

- For each iteration, larger element move up (or smaller element move down).

```
BubbleSort(A)
{
  for j = |A|-1 down to 0 do
  {
      for k=2 to j do
      {
          If A[k-1] > A[k]
              Swap (A[k-1], A[k])
      }
  }
}
```

C_1
C_2
C_3
C_4

Running Time Analysis with Sorting Algorithms
(Bubble Sort: Best Case)

- For each iteration, larger element move up (or smaller element move down).

```
BubbleSort(A)
{
  for j = |A|-1 down to 0 do
  {
      for k=2 to j do
      {
          If A[k-1] > A[k]
              Swap (A[k-1], A[k])
      }
  }
}
```

$C_1 n$
$C_2(n-1)$
$C_3 \frac{n(n-1)}{2}$

Running Time Analysis with Sorting Algorithms
(Bubble Sort: Best Case)

Bubble up	1	2	3	4	5	6
Bubble up	1	2	3	4	5	6
Bubble up	1	2	3	4	5	6
Bubble up	1	2	3	4	5	6
Bubble up	1	2	3	4	5	6
	1	2	3	4	5	6

Unsorted sub list
Sorted sub list

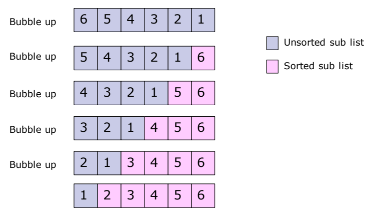
Running Time Analysis with Sorting Algorithms
(Bubble Sort: Worst Case)

- For each iteration, larger element move up (or smaller element move down).

```
BubbleSort(A)
{
  for j = |A|-1 down to 0 do
  {
      for k=2 to j do
      {
          If A[k-1] > A[k]
              Swap (A[k-1], A[k])
      }
  }
}
```

$C_1 n$
$C_2(n-1)$
$C_3 \frac{n(n-1)}{2}$
$C_4 \frac{n(n-1)}{2}$

Running Time Analysis with Sorting Algorithms
(Bubble Sort: Worst Case)



Running Time Analysis with Sorting Algorithms
(Bubble Sort)

Bubble Sort Analysis

- Best Case:

$$T(n) = C_1 n + C_2(n-1) + C_3\frac{n(n-1)}{2}$$
$$= \frac{C_3}{2}n^2 + \left(C_1 + C_2 - \frac{C_3}{2}\right)n - C_2 = \Theta(n^2)$$

- Worst Case:

$$T(n) = C_1 n + C_2(n-1) + C_3\frac{n(n-1)}{2} + C_4\frac{n(n-1)}{2}$$
$$= \left(\frac{C_3}{2} + \frac{C_4}{2}\right)n^2 + \left(C_1 + C_2 - \frac{C_3}{2} - \frac{C_4}{2}\right)n - C_2 = \Theta(n^2)$$

Running Time Analysis with Sorting Algorithms
(Shell Sort)

- Shell sort is a simple extension of insertion sort that gains speed by allowing exchanges of elements that are far apart.
- The speed of shell sort is depending on the chose of sequence of h (sequence of apart)

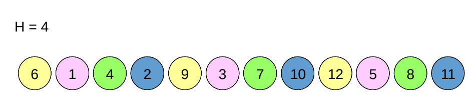
Running Time Analysis with Sorting Algorithms
(Shell Sort)

- Shell sort is a simple extension of insertion sort that gains speed by allowing exchanges of elements that are far apart.
- Array is divided into group of shell.
- Call insertion sort for each shell.
- The speed of shell sort is depending on the choice of sequence of h (sequence of apart)

Running Time Analysis with Sorting Algorithms
(Shell Sort)

```
void ShellSort (A, int  h[])
{
   for each h[i]
   {
      Divide array into h shell group
      Call insertion Sort for each shell group;
   }
}
```


Running Time Analysis
(Shell Sort)



Running Time Analysis
(Shell Sort)

H = 2

4 1 6 2 7 3 8 5 9 10 12 11

Running Time Analysis
(Shell Sort)

H = 1

1 2 3 4 5 6 7 8 9 10 11 12

Running Time Analysis
(Shell Sort)

- The speed of shell sort is depending on the chose of sequence of h Theorem)

The running time of Shell sort is O($n^{3/2}$) for the increments of h 1, 4, 13, 40, 121, 364, 1093, 3280, 9841, …).
Theorem)
The running time of Shell sort is O($n^{4/3}$) for the increments of h 1, 8, 23, 77, 281, 1073, 4193, 16577, 9841, …).

Summery of Sorting algorithm Analysis

Algorithm	Best Case	Worst Case	Average Case
Insertion Sort	O(n)	O(n^2)	O(n^2)
Selection Sort	O(n^2)	O(n^2)	O(n^2)
Bubble Sort	O(n^2)	O(n^2)	O(n^2)
Shell Sort	O(n^k) (1< k <2)	O(n^k) (1< k <2)	O(n^k) (1< k <2)