Sorting Algorithm

Presenters :

Amirreza Tavakoli

And

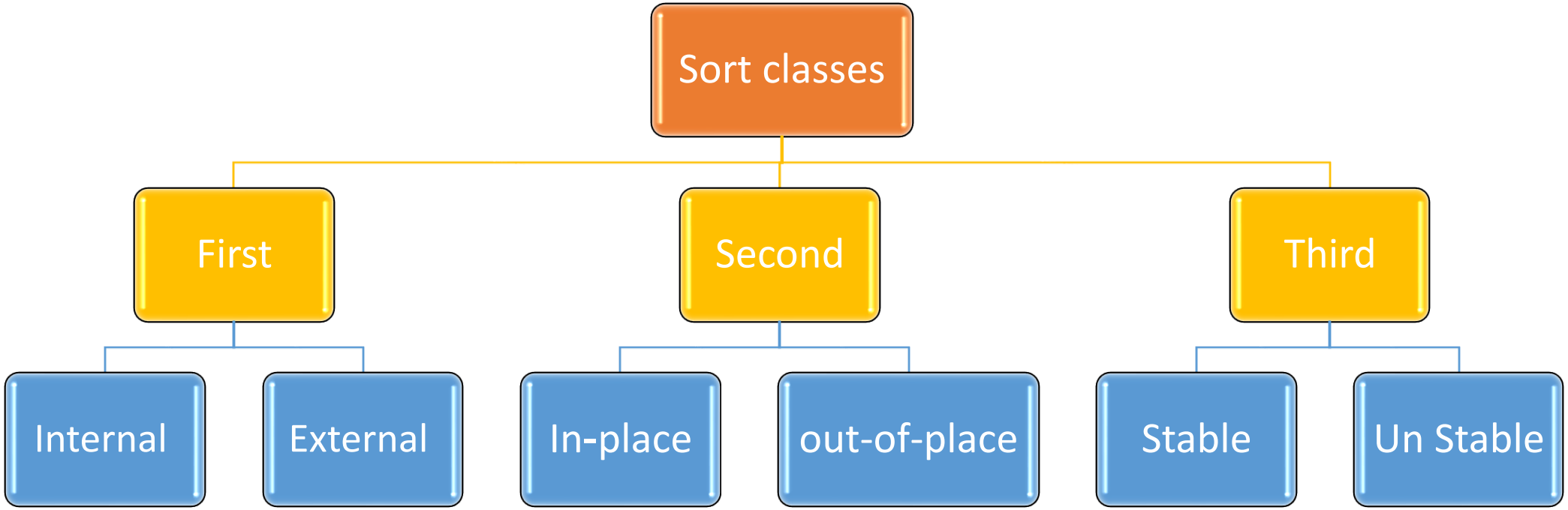
Behnia Soleimani

Produced by :

Amirreza Tavakoli

Contents

Introduction



Internal Sorting

- An internal sort is any data sorting process that takes place entirely within the main memory of a computer.

- Some common internal sorting algorithms :
  - Bubble Sort
  - Insertion Sort
  - Quick Sort
  - Heap Sort
  - Radix Sort
  - Selection sort

External Sorting

- External sorting is a class of sorting algorithms that can handle massive amounts of data.
- External sorting is required when the data being sorted do **not fit into the main memory** of a computing device (usually RAM) and instead they must reside in the **slower external memory**, usually a hard disk drive

In-place Sorting

- In computer science, an in-place algorithm is an algorithm which transforms input using no auxiliary data structure.

```
template<class T>
void InplaceSwap(T& input1 , T& input2)
{
    input1 = input1 + input2;
    input2 = input1 - input2;
    input1 = input1 - input2;
}
```

See C# Code

See C++ Code

out-of-place Sorting

- An algorithm which is not in-place is sometimes called not-in-place or out-of-place.

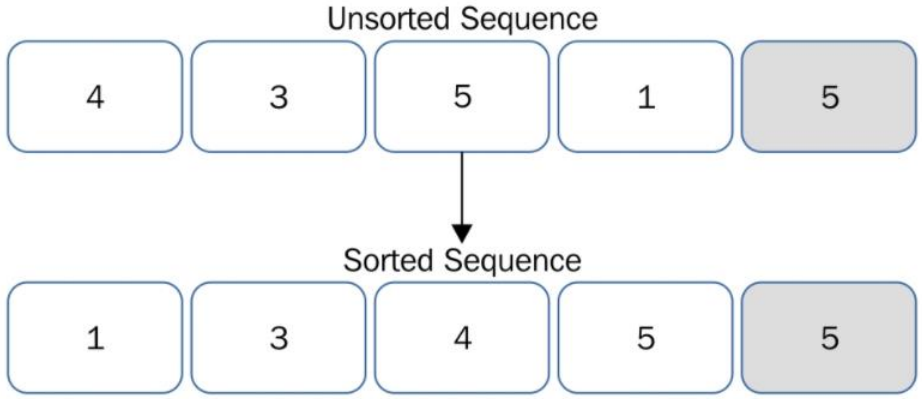
```
template<class T>
void OutofplaceSwap(T& input1 , T& input2)
{
    T Auxiliary;
    Auxiliary = input1;
    input1 = input2;
    input2 = Auxiliary;
}
```


See C# Code


See C++ Code

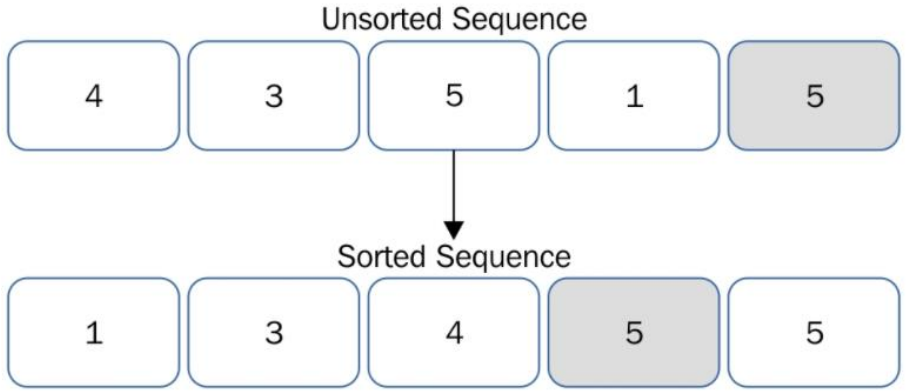
Stable

A sorting algorithm is said to be stable if two objects with equal keys appear in the same order in sorted output as they appear in the input unsorted array.



Un Stable

Unstable sorting algorithms do not maintain the relative ordering of elements of equal values in a sorted sequence



نکته

- تمامی الگوریتم های نامتعادل مرتب سازی را می توان بدون افزایش مرتبه زمانی متعادل کرد .(با لحاظ کردن اندیس ها)

Early Humans Sorting

Review – Selection Sort

The algorithm maintains two subarrays in a given array:
1) The subarray which is already sorted.
2) Remaining subarray which is unsorted.

In every iteration of selection sort, the minimum element (considering ascending order) from the unsorted subarray is picked and moved to the sorted subarray.
Initially, the sorted part is empty and the unsorted part is the entire list

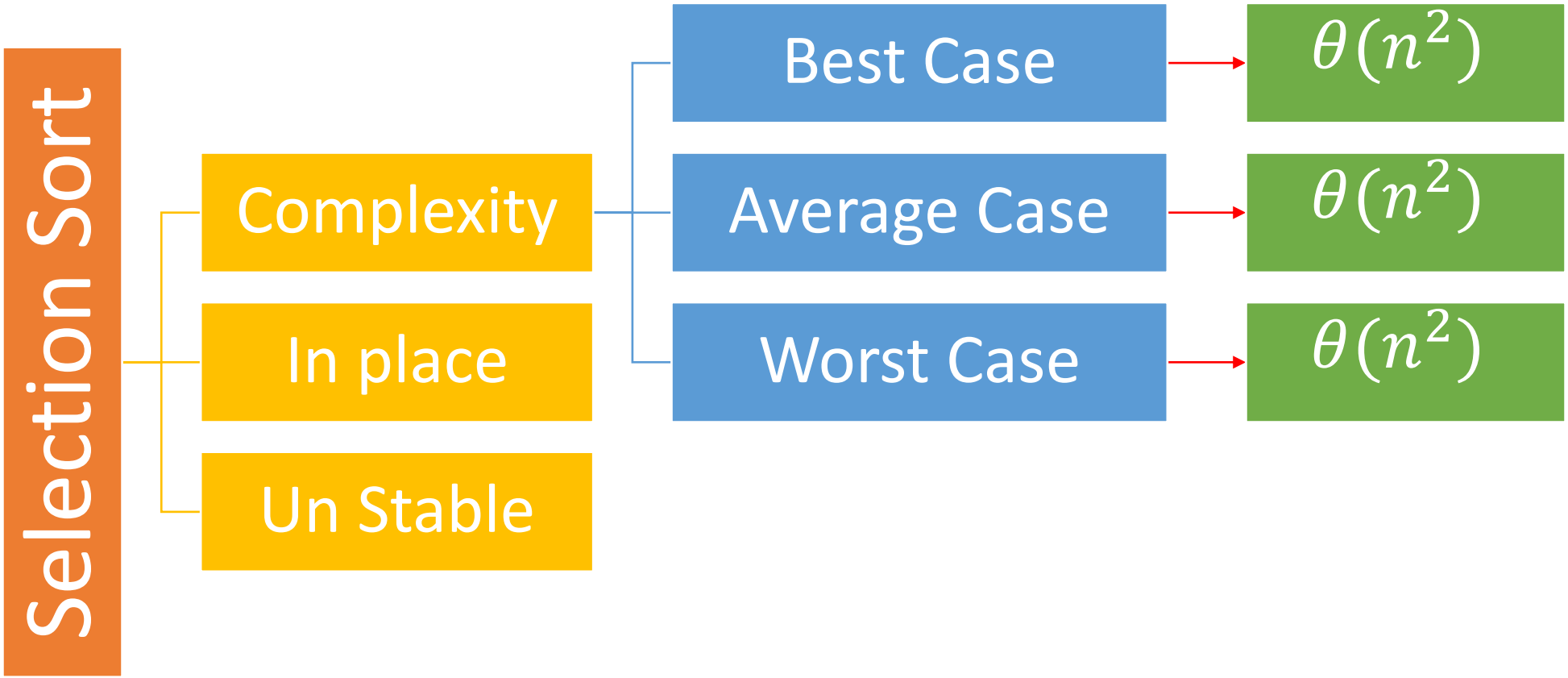
Review – Selection Sort - Pseudocode

```
   list  : array of items
   n     : size of list

   for i = 1 to n - 1
   /* Step 1 - set current element as minimum */
      min = i

      /* Step 2 – Search the minimum element in the list */
      for j = i+1 to n
         if list[j] < list[min] then
            min = j;
         end if
      end for

      /* Step 3 – Swap with value at location MIN */
      if  min  != i  then
         swap list[min] and list[i]
      end if
      /* Step 4 – Increment MIN to point to next element */
      /* Step 5 – Repeat until list is sorted */
   end for

end procedure
```

See C# Code

See C++ Code

Review – Selection Sort – Details



Review – Bubble Sort

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

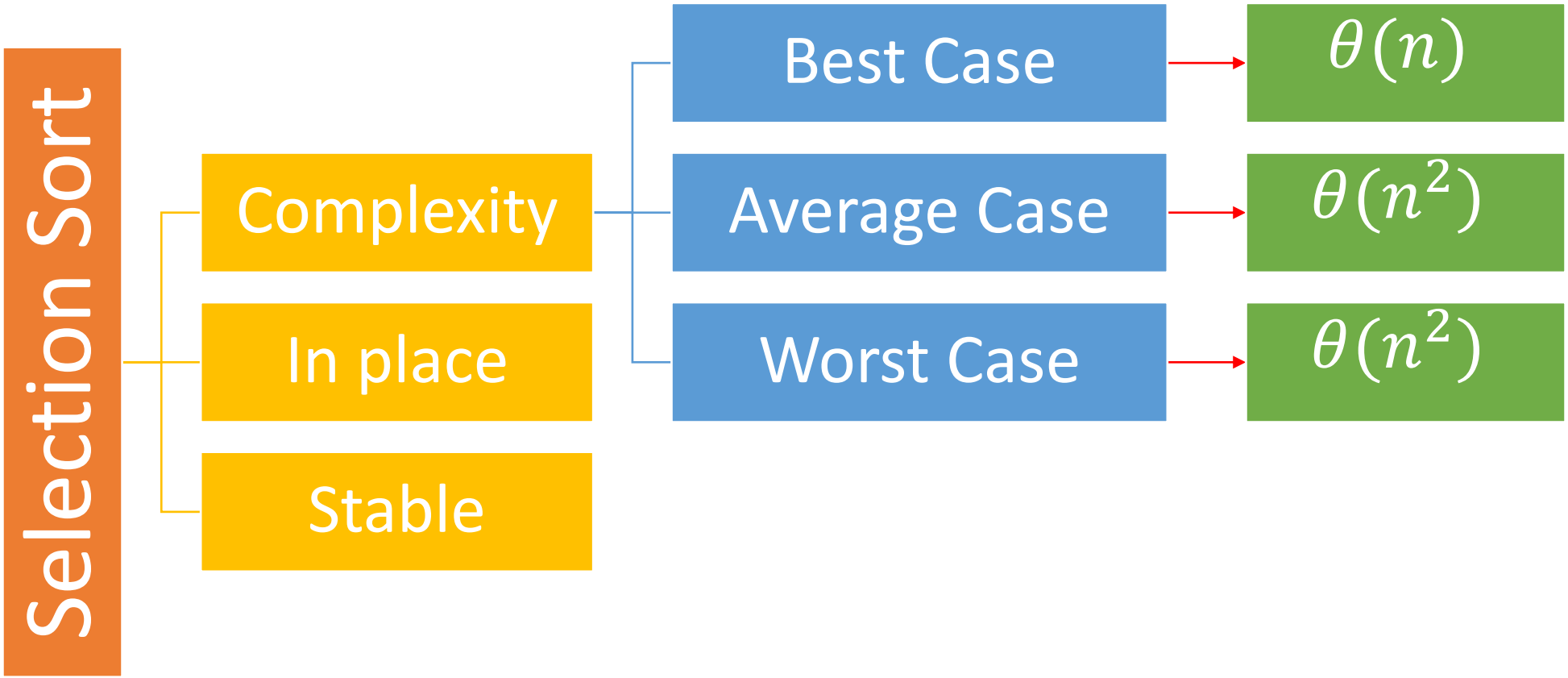
Review – Bubble Sort - Pseudocode

```
    list  : array of items
    n     : size of list
    flag = true
    for i = 1 to n - 1

       flag = false

       for j = 1 to n-i-1

          if list[j] > list[j+1] then
              swap  list[j] and list[j+1]
              flag = true
          end if

       end for

       if  !flag then
          break;
       end if

    end for

end procedure
```


See C# Code


See C++ Code

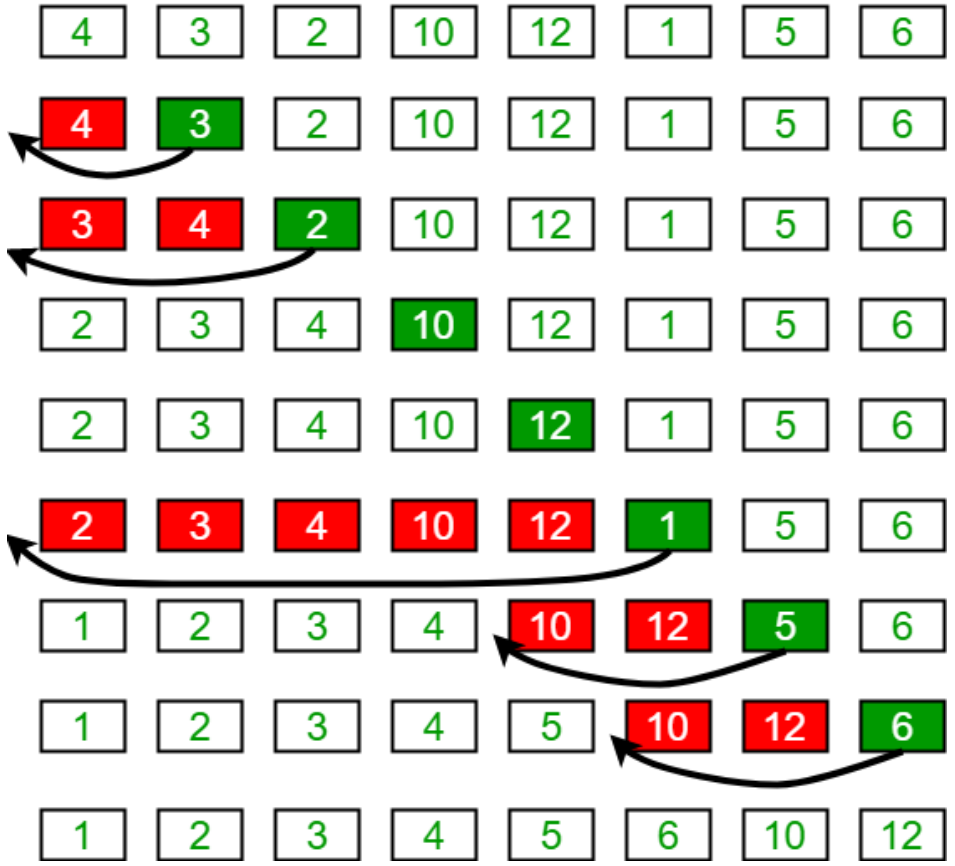
Review – Bubble Sort – Details



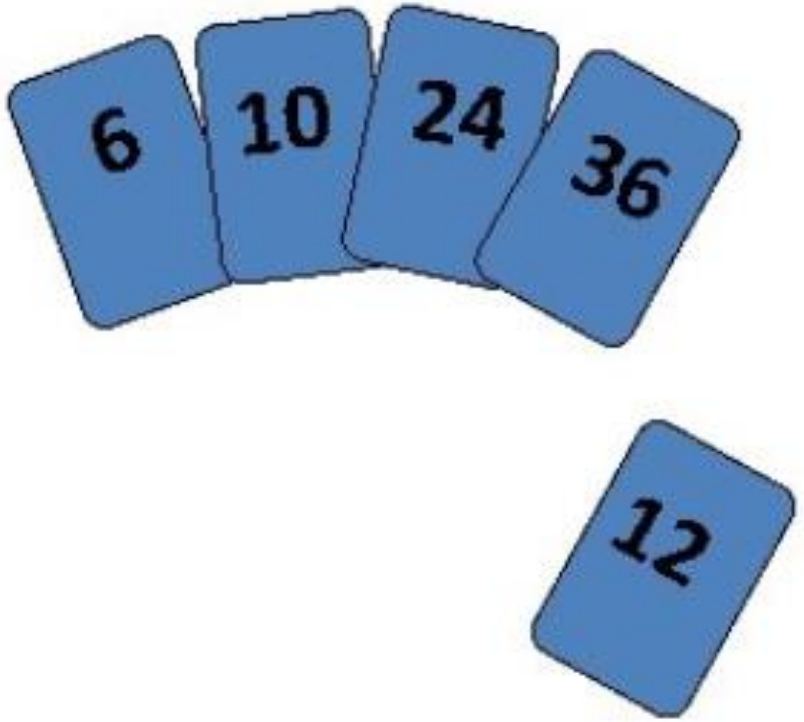
Review – Insertion Sort

Insertion sort is a simple sorting algorithm that works similar to the way you sort playing cards in your hands.
The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.

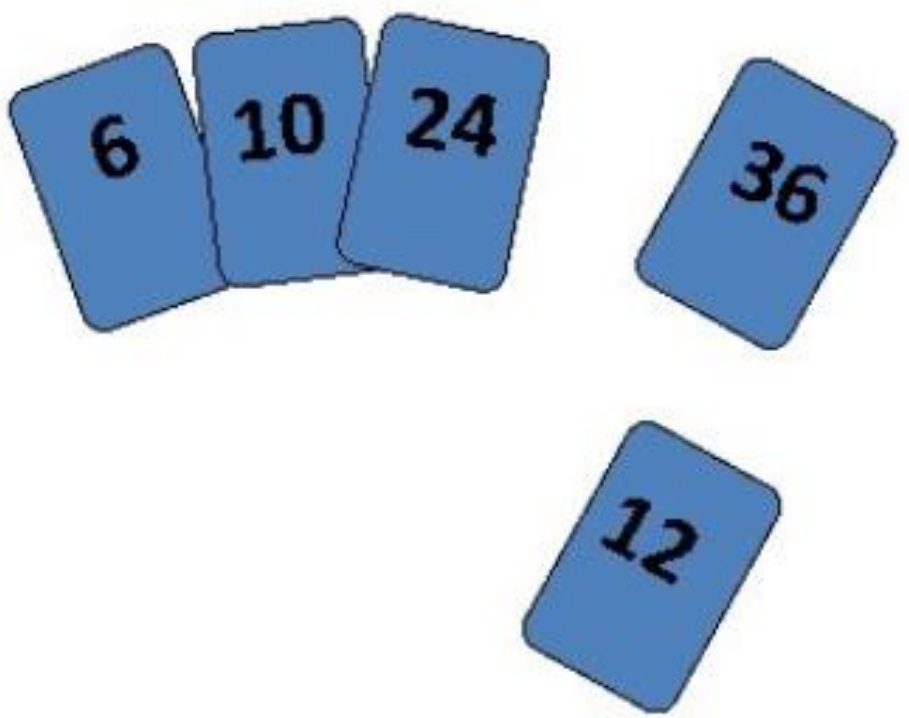
Review – Insertion Sort



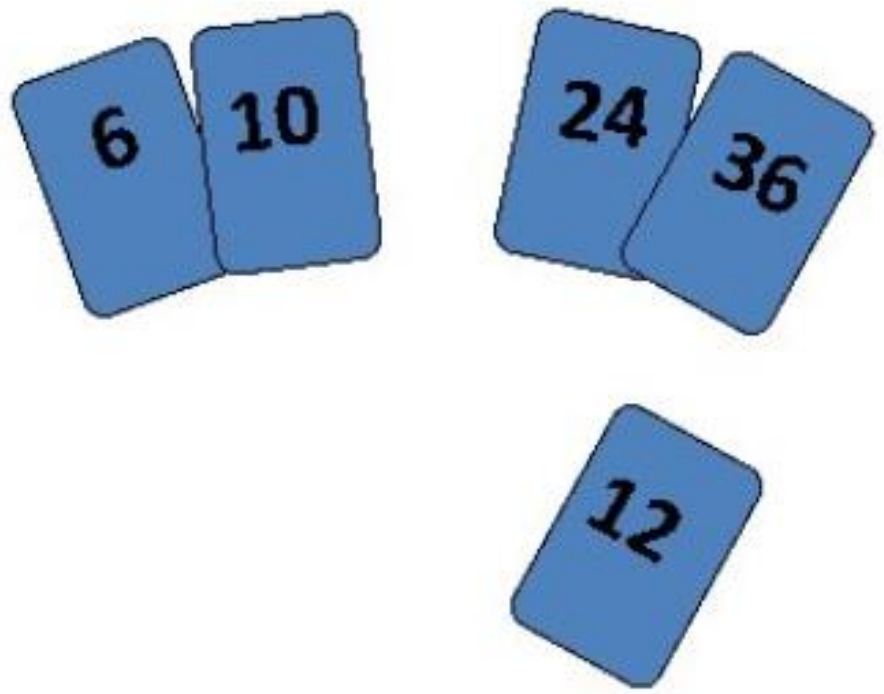
Review – Insertion Sort



Review – Insertion Sort



Review – Insertion Sort



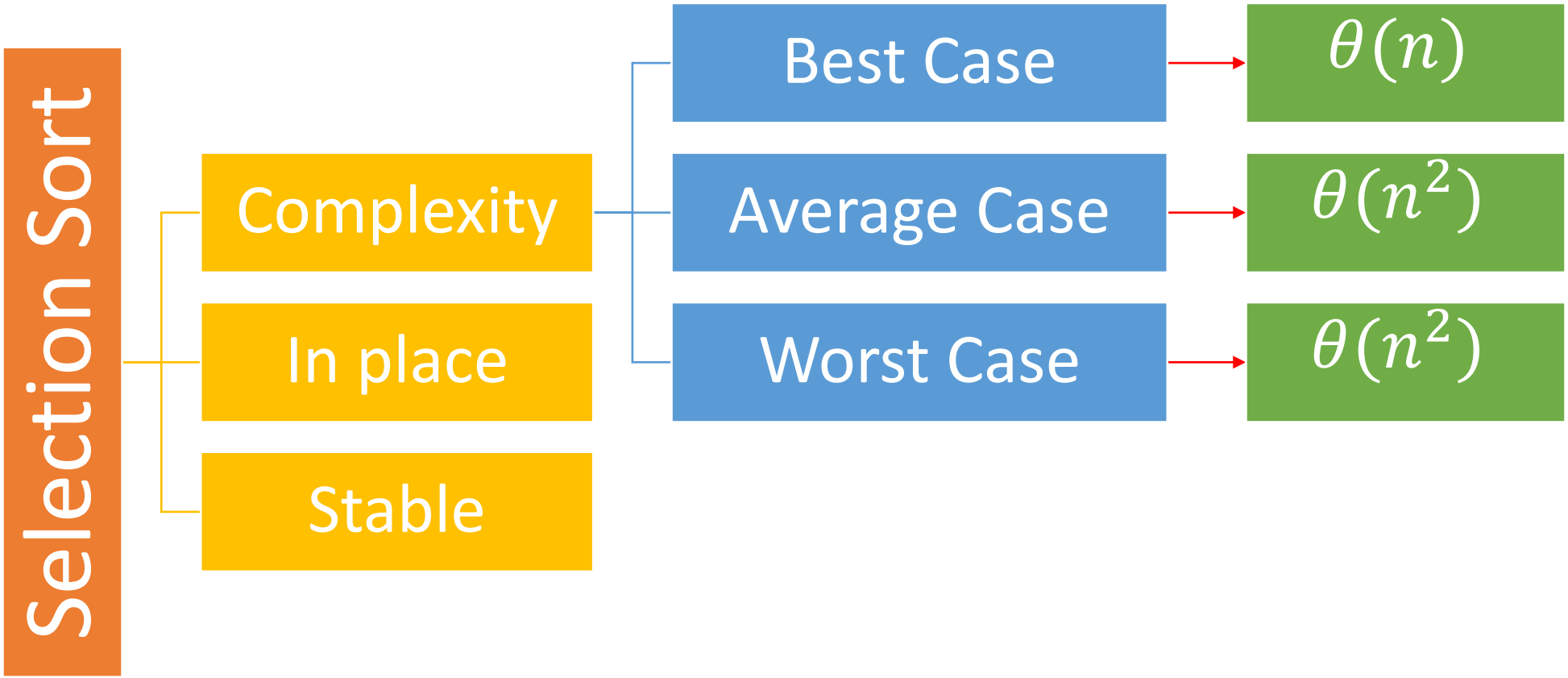
Review – Insertion Sort - Pseudocode

```
   list  : array of items
   n     : size of list

   for i = 2 to n
      x = list[i]
      j = i-1

    while (list[j] > x && j > 0)
           list[j+1] =  list[j]
           j--
      end for

       list[j+1] =  y

   end for

end procedure
```


See C# Code

See C++ Code

Review – Insertion Sort – Details



Modern Humans Sorting

Data Structure - Spring 2021

Review – Merge Sort

- Merge Sort is a Divide and Conquer algorithm.
- It divides the input array into two halves, calls itself for the two halves, and then merges the two sorted halves.
- The merge() function is used for merging two halves.

Review – Merge Sort

MergeSort(arr[], l, r)
If r > 1

1. Find the middle point to divide the array into two halves:
   middle m = l+ (r-l)/2
2. Call mergeSort for first half:
   Call mergeSort(arr, l, m)
3. Call mergeSort for second half:
   Call mergeSort(arr, m+1, r)
4. Merge the two halves sorted in step 2 and 3:
   Call merge(arr, l, m, r)

Review – Merge Sort

2	4	6	8	10
1	3	5	7	9

نکته

- تعداد مقایسه های ادغام لیست مرتب m , n عنصری حداکثر m+n-1 تاست.(حداقل{min{m,n)
- تعداد مقایسه های ادغام لیست مرتب $\left\lfloor \frac{n}{2} \right\rfloor, \left\lceil \frac{n}{2} \right\rceil$ عنصری حداکثر n-1 تاست.(حداقل $\left\lfloor \frac{n}{2} \right\rfloor$)
- ادغام لیست مرتب $\left\lfloor \frac{n}{2} \right\rfloor, \left\lceil \frac{n}{2} \right\rceil$ از مرتبه $\theta(n)$ می باشد

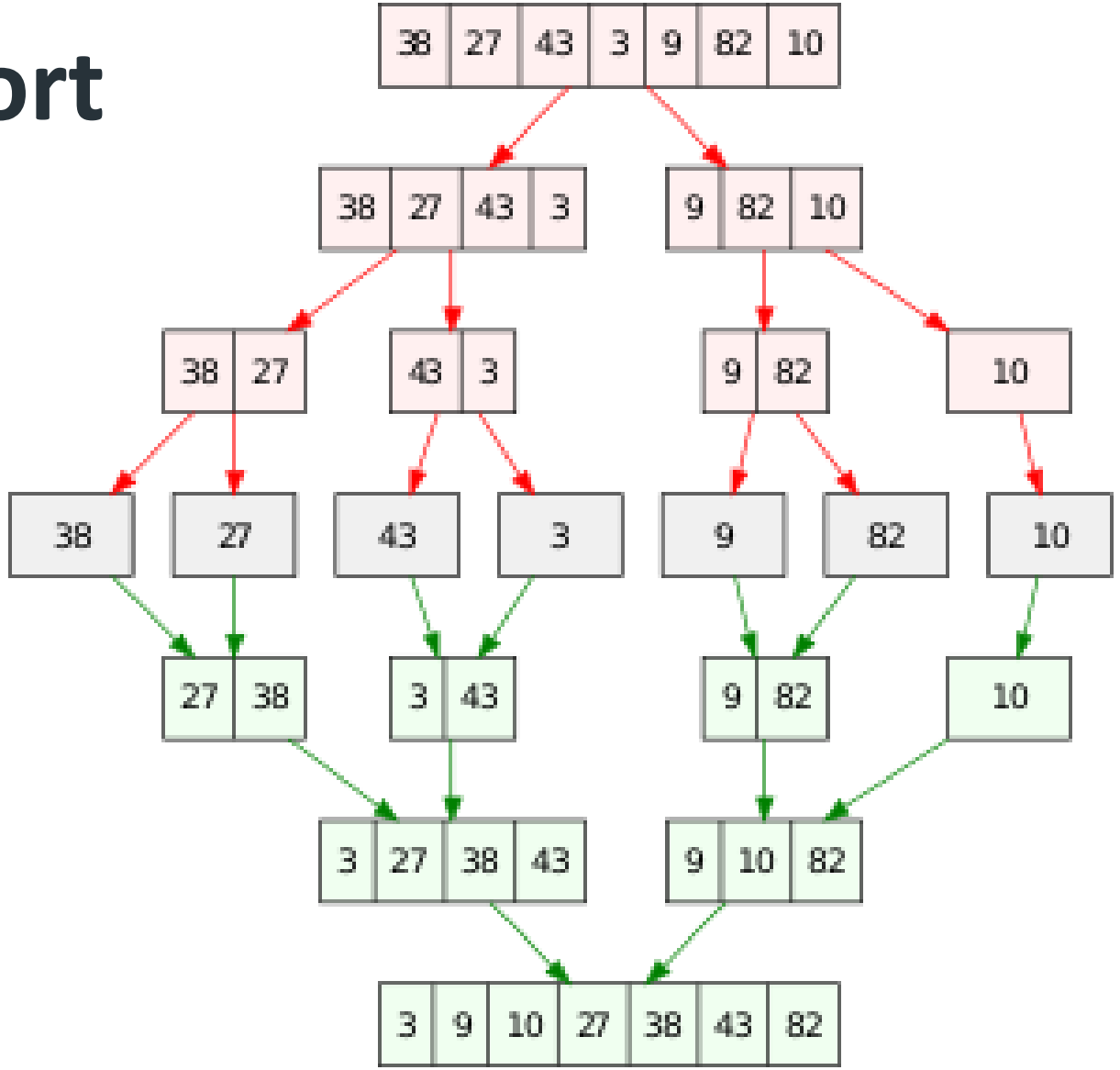
Review – Merge Sort

The following diagram from wikipedia shows the complete merge sort process for an example array {38, 27, 43, 3, 9, 82, 10}

If we take a closer look at the diagram, we can see that the array is recursively divided in two halves till the size becomes 1. Once the size becomes 1, the merge processes come into action and start merging arrays back till the complete array is merged

Review – Merge Sort

$$T(n) = T\left(\left\lfloor\frac{n}{2}\right\rfloor\right) + T\left(\left\lceil\frac{n}{2}\right\rceil\right) + \theta(n)$$



Review – Merge Sort - Pseudocode

```
static void sort(int[] arr, int low, int up)
{
        if (low < up)
        {
                int mid = (low + up) / 2;

                sort(arr, low, mid);
                sort(arr, mid + 1, up);

                merge(arr, low, mid, up);
        }
}
```

See C# Code

See C++ Code

Review – Merge Sort - Pseudocode

```
static void merge(int[] arr, int low, int mid, int up)
    {

        int lenL = mid - low + 1;
        int lenR = up - mid;

        int[] L = new int[lenL];
        int[] R = new int[lenR];

        int i = 0;
        int j = 0;
        int k = low;

        Array.Copy(arr , low , L , 0 , lenL);
        Array.Copy(arr , mid + 1 ,  R , 0 , lenR);

        while (i < lenL && j < lenR)
        {
            if (L[i] <= R[j])
                arr[k++] = L[i++];
            else
                arr[k++] = R[j++];
        }

        while (i < lenL)
            arr[k++] = L[i++];

        while (j < lenR)
            arr[k++] = R[j++];
    }
```


See C# Code

See C++ Code

سوال

- برای آرایه ۱۰ عنصری
  - Merge sort چند بار فراخوانی می شود ؟
  - Merge چند بار فراخوانی می شود؟
  - حداکثر تعداد مقایسه ها ؟

نکته

- تعداد فراخوانی در Merge sort ؟
- Merge چند بار فراخوانی می شود؟
- حداکثر تعداد مقایسه های Merge؟

$T(n) = T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + 1 = 2n - 1$ (T(1) = 1 , T(2) = 3)

$T(n) = T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + 1 = n - 1$ (T(1) = 0 , T(2) =1)

$T(n) = T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + n - 1 = nlogn - n + 1$

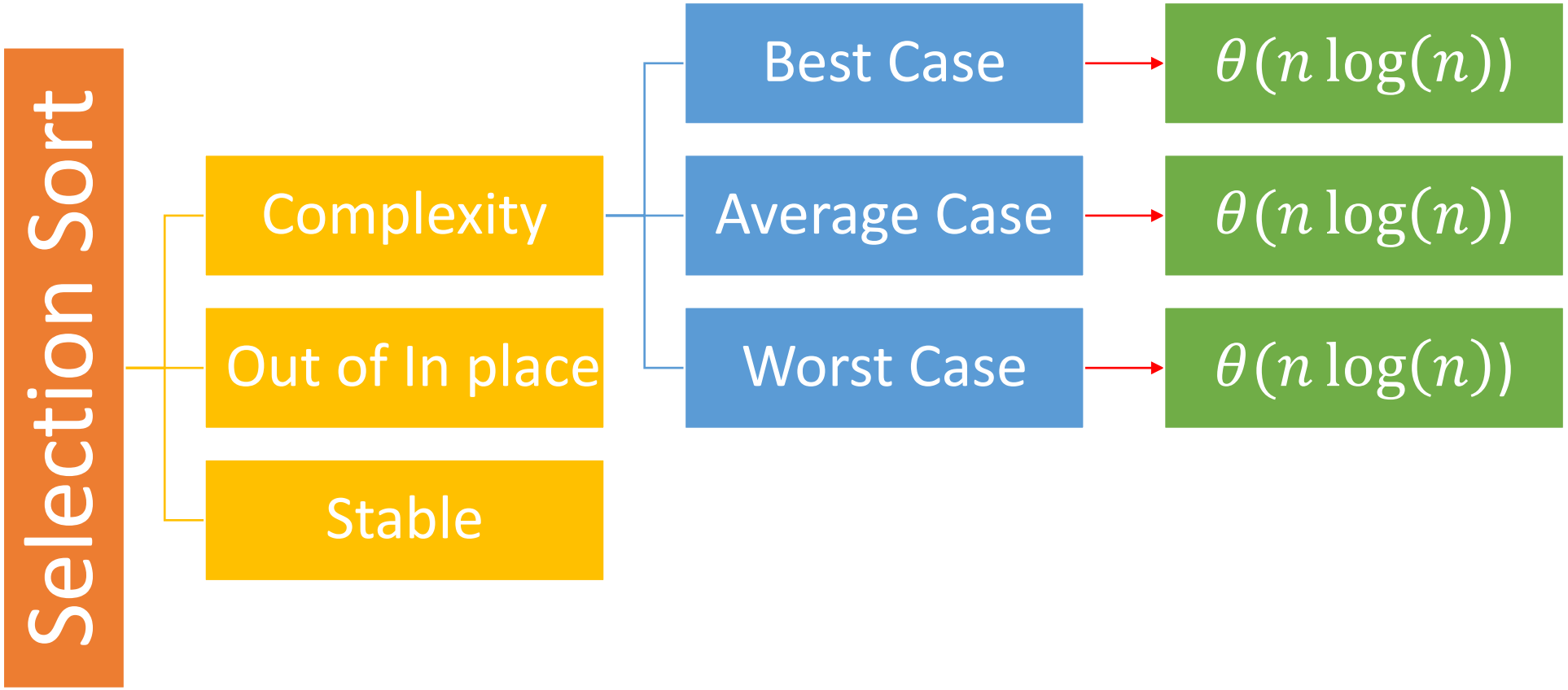
سوال

- در مرتب سازی ادغامی فرض کن آرایه را آنقدر نصف کردیم که به لیست های K عنصری رسیدیم سپس این لیست های k عنصری را با روش درجی مرتب کرده ایم و سپس شروع به ادغام کردیم ،مرتبه زمانی را بر حسب k,n بیابید.

سوال

- ادغام K تا لیست مرتب $\frac{n}{k}$ عنصری چقدر زمان می برد اگر
  - ابتدا ۱ با ۲ سپس نتیجه اش با ۳ و ... ادغام کنیم ؟
  - لیست ها را دو به دو ادغام کنیم ؟
  - ار هرم کمینه برای ادغام کمک بگیریم؟

Review - Merge Sort – Details



نکته

- اگر تعداد عناصر آرایه کم باشد یا آرایه از پیش مرتب شده باشد بهترین روش درجی است.
- اگر عناصر آرایه مساوی باشند روش های حبابی ، درجی و هرمی از مرتبه $\theta(n)$ هستند.
- اگر آرایه از قبل مرتب باشد (در جهت الگوریتم) روش های حبابی و درجی $\theta(n)$ هستند.
- روش های حبابی و درجی و سریع حداکثر $\frac{n(n-1)}{2}$ مقایسه دارند ولی روش انتخابی دقیقاً $\frac{n(n-1)}{2}$ مقایسه دارد.
- اگر مقایسه هزینه نداشته باشد ولی جا به جایی هزینه زیادی داشته باشد بهترین روش انتخابی است چون تعداد جا به جایی در آن از همه روش ها کمتر است.
- اگر آرایه مرتب باشد روش های سریع و درختی بدترین حالت را دارند.

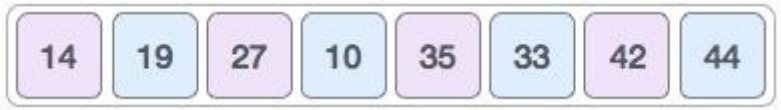
Shell Sort

Shell sort is a highly efficient sorting algorithm and is based on insertion sort algorithm.

In insertion sort, we move elements only one position ahead.

When an element has to be moved far ahead, many movements are involved. The idea of shellSort is to allow exchange of far items.

How Shell Sort Works?



Shell Sort – C++ Code

```
int shellSort(int arr[], int arrLen)
{
    for (int gap = arrLen/2; gap > 0; gap /= 2)
    {
        for (int i = gap; i < arrLen; i += 1)
        {
            int temp = arr[i];
            int j = i - gap;
            while (arr[j] > temp && j >= gap)
            {
                arr[j + gap] = arr[j];
                j = j - gap;
            }
            arr[j] = temp;
        }
    }
    return 0;
}
```

See C# Code

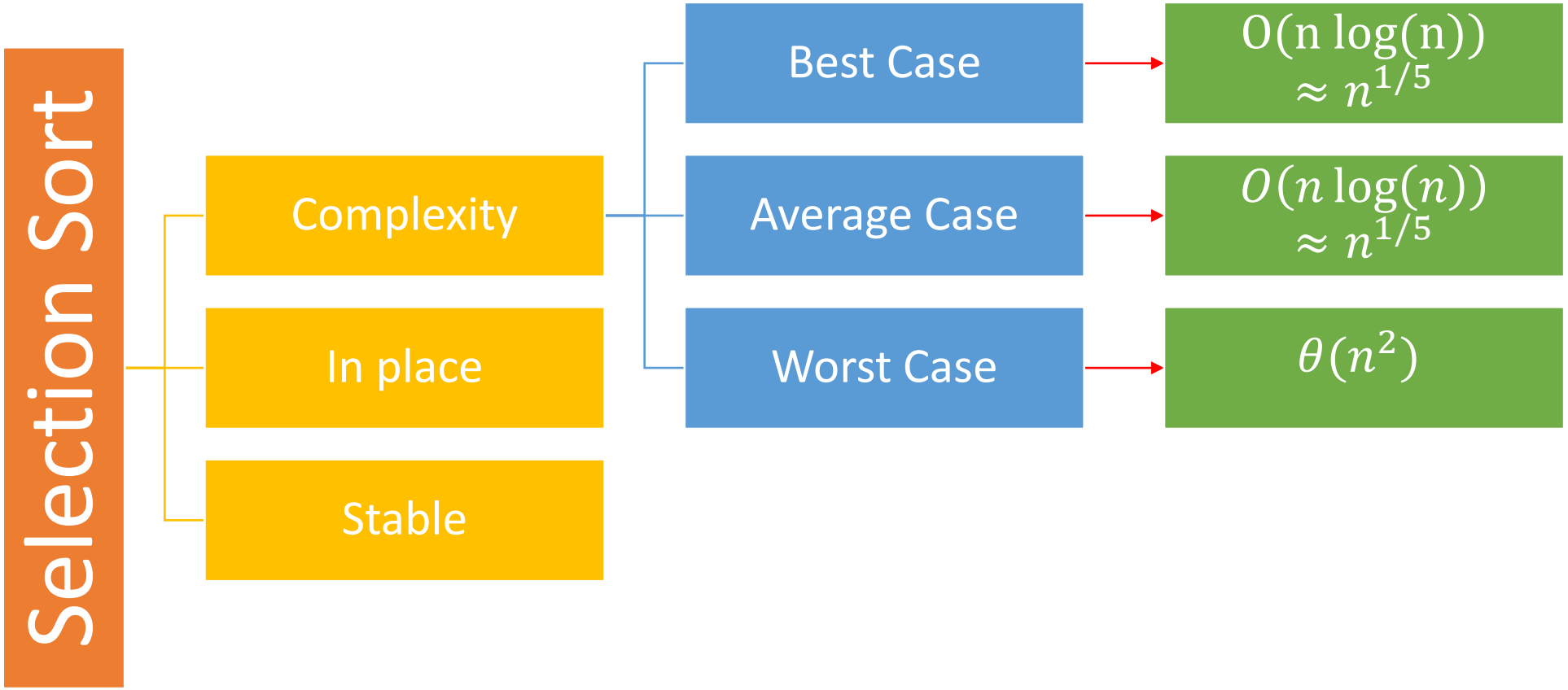
See C++ Code

Self Check

Given the following list of numbers: [5, 16, 20, 12, 3, 8, 9, 17, 19, 7] Which answer illustrates the contents of the list after all swapping is complete for a gap size of 2?

Index	0	1	2	3	4	5	6	7	8	9
Value	5	16	20	12	3	8	9	17	19	7

Shell Sort – Details



Algorithm Sort For String in cpp

```cpp
bool compare(string x, string y)
{
    char first;
    char second;
    for (size_t i = 0; i < min(x.length() , y.length()); i++)
    {
        first = tolower(x[i]);
        second = tolower(y[i]);
        if (first == second)
            continue;
        else if (first > second)
            return true;
        else
            return false;
    }
    return true;

}
```

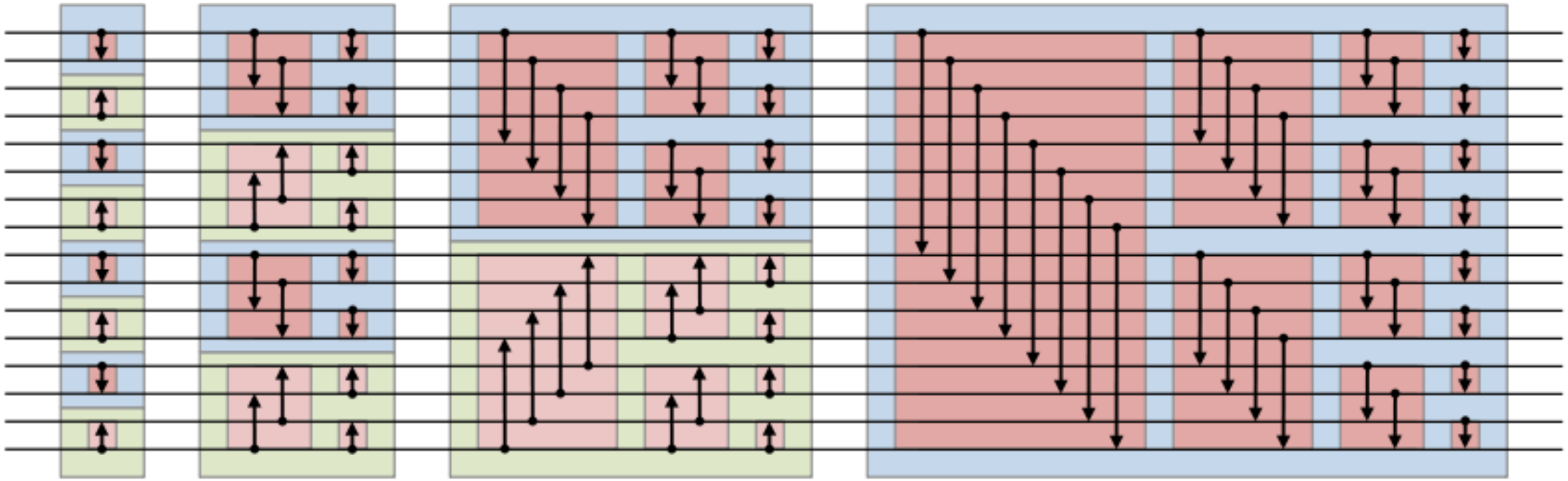

See C++ Code

Algorithm Sort For String in cpp

```
bool compare(string x, string y)
    {
        int min(int a, int b) => ((a > b) ? b : a);
        x = x.ToLower();
        y = y.ToLower();
        for (int i = 0; i < min(x.Length, y.Length); i++)
        {
            if (x[i] == y[i])
                continue;
            else if (x[i] > y[i])
                return true;
            else
                return false;
        }
        return true;
    }
```

See C# Code

Algorithm	Best Time Complexity	Average Time Complexity	Worst Time Complexity	Worst Space Complexity
Linear Search	O(1)	O(n)	O(n)	O(1)
Binary Search	O(1)	O(log n)	O(log n)	O(1)
Bubble Sort	O(n)	O(n^2)	O(n^2)	O(1)
Selection Sort	O(n^2)	O(n^2)	O(n^2)	O(1)
Insertion Sort	O(n)	O(n^2)	O(n^2)	O(1)
Merge Sort	O(nlogn)	O(nlogn)	O(nlogn)	O(n)
Quick Sort	O(nlogn)	O(nlogn)	O(n^2)	O(log n)
Heap Sort	O(nlogn)	O(nlogn)	O(nlogn)	O(n)
Bucket Sort	O(n+k)	O(n+k)	O(n^2)	O(n)
Radix Sort	O(nk)	O(nk)	O(nk)	O(n+k)
Tim Sort	O(n)	O(nlogn)	O(nlogn)	O(n)
Shell Sort	O(n)	O((nlog(n))^2)	O((nlog(n))^2)	O(1)



Parallel Sorting

Data Structure - Spring 2021

Bitonic Sort

Bitonic sort is a comparison-based sorting algorithm that can be run in parallel

Bitonic sort does O(n Log ^{2}n) comparisons.

The number of comparisons done by Bitonic sort are more than popular sorting algorithms like Merge Sort [does O(nLogn) comparisons], but Bitonice sort is better for parallel implementation because we always compare elements in predefined sequence and the sequence of comparison doesn't depend on data.

Therefore it is suitable for implementation in hardware .

Bitonic Sort must be done if number of elements to sort are 2^n. The procedure of bitonic sequence fails if the number of elements are not in aforementioned quantity precisely.

To understand Bitonic Sort, we must first understand what is Bitonic Sequence and how to make a given sequence Bitonic.

Bitonic Sequence

In order to understand Bitonic sort, we must understand Bitonic sequence. Bitonic sequence is the one in which, the elements first comes in increasing order then start decreasing after some particular index. An array A[0... i ... n-1] is called Bitonic if there exist an index i such that,

$$A[0] < A[1] < A[2] .... A[i-1] < A[i] > A[i+1] > A[i+2] > A[i+3] > ... > A[n-1]$$

How to convert Random sequence to Bitonic sequence ?

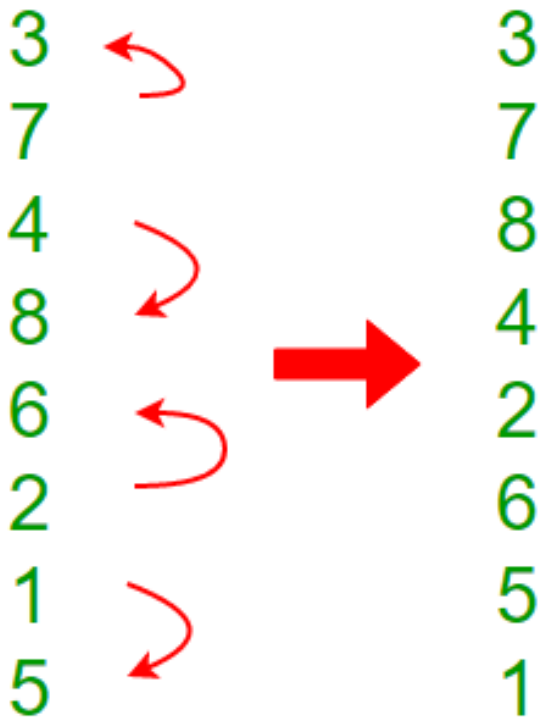
We start by forming 4-element bitonic sequences from consecutive 2-element sequence. Consider 4-element in sequence x0, x1, x2, x3. We sort x0 and x1 in ascending order and x2 and x3 in descending order. We then concatenate the two pairs to form a 4 element bitonic sequence.
Next, we take two 4 element bitonic sequences, sorting one in ascending order, the other in descending order (using the Bitonic Sort which we will discuss below), and so on, until we obtain the bitonic sequence.

How to convert Random sequence to Bitonic sequence ?

Convert the following sequence to bitonic sequence: 3, 7, 4, 8, 6, 2, 1, 5

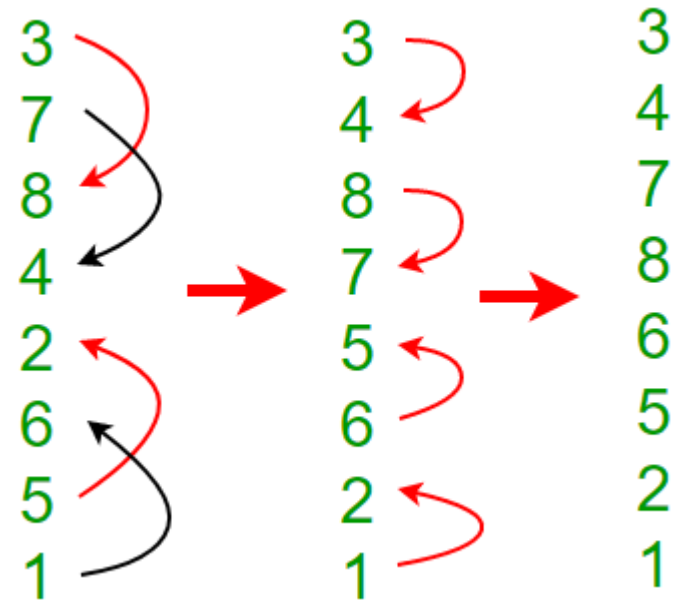
Step 1: Consider each 2-consecutive elements as bitonic sequence and apply bitonic sort on each 2- pair elements. In next step, take two 4 element bitonic sequences and so on.

How to convert Random sequence to Bitonic sequence ?



How to convert Random sequence to Bitonic sequence ?

Note: x0 and x1 are sorted in ascending order and x2 and x3 in descending order and so on
Step 2: Two 4 element bitonic sequences : **A**(3,7,8,4) and **B**(2,6,5,1) with comparator length as 2

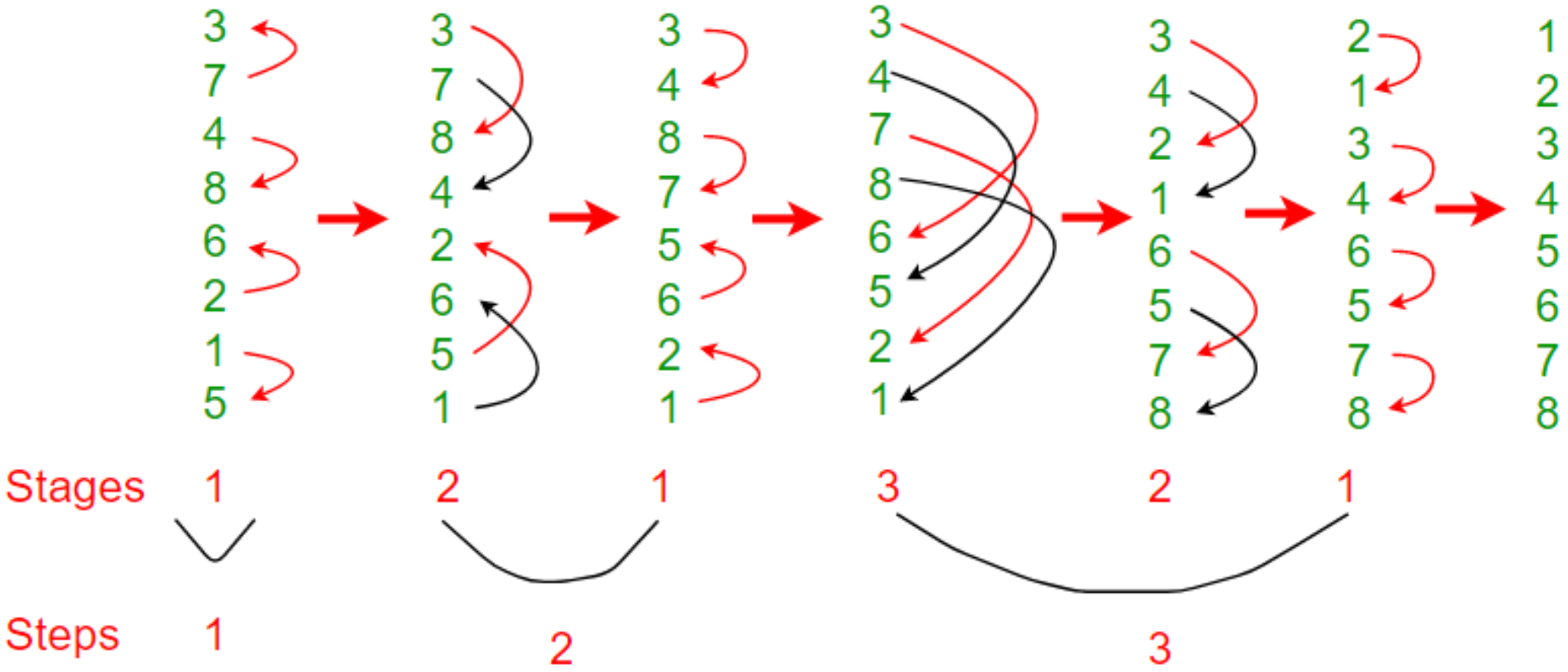


How to convert Random sequence to Bitonic sequence ?

After this step, we'll get Bitonic sequence of length 8.

3, 4, 7, 8, 6, 5, 2, 1

Bitonic Sorting



Bitonic Sort – C# Code

```
public static void bitonicSort(int[] a, int low, int count, Direction dir)
    {
        if (count > 1)
        {
            int k = count / 2;

            bitonicSort(a, low, k, Direction.ascending);

            bitonicSort(a, low + k, k, Direction.descending);

            bitonicMerge(a, low, count, dir);
        }
    }
```

See C# Code

See C++ Code

Bitonic Sort – C# Code

```
public static void bitonicMerge(int[] a, int low, int count, Direction dir)
    {
        if (count > 1)
        {
            int k = count / 2;
            for (int i = low; i < low + k; i++)
                compAndSwap(a, i, i + k, dir);

            bitonicMerge(a, low, k, dir);
            bitonicMerge(a, low + k, k, dir);
        }
    }
```


See C# Code

See C++ Code

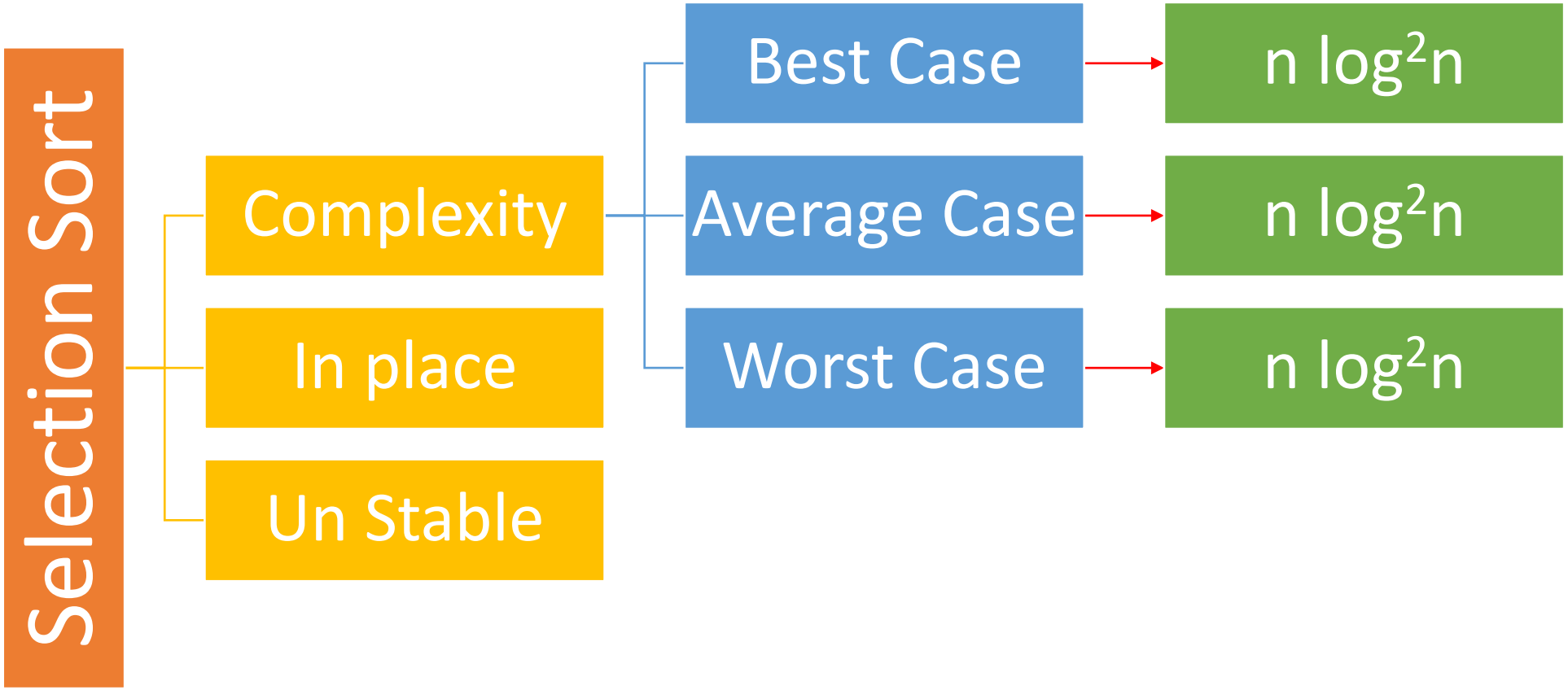
Bitonic Sort – C# Code

```
public static void compAndSwap(int[] a, int i, int j, Direction dir)
    {
        if ((a[i] > a[j]))
        {
            if (dir == Direction.ascending)
            {
                Swap(ref a[i], ref a[j]);
            }
        }
        else if(dir == Direction.descending)
        {
            Swap(ref a[i], ref a[j]);
        }

    }
```


See C# Code

See C++ Code

Bitonic Sort – Details



Time Elapsed

Length	Bitonic	Insertion	Shell	Selection	Bubble	Merge
8 2^3	00:00.0004034	00:00.0001264	00.0000993	00:00.0001317	00:00.0001423	00:00.0003212
64 2^6	00:00.0004182	00:00.0001305	00:00.0001047	00:00.0001398	00:00.0001502	00:00.0003465
512 2^9	00:00.0010683	00:00.0004652	00:00.0002394	00:00.0006698	00:00.0010079	00:00.0004115
16384 2^{14}	00:00.0226264	00:00.1783594	00:00.0057087	00:00.2990532	00:00.7612952	00:00.0039391
65536 2^{16}	:00:02.0610621	00:03.8383104	00:00.0360226	00:11.4131500	00:14.8394458	00:00.0150015


See C# Code


See C++ Code

درخت تصمیم

- درختی دو دویی که برای الگوریتم های مقایسه ای کشیده می شود و گره های داخلی مقایسه دو عنصر را نشان می دهند . یال چپ یعنی عنصر اول کوچکتر است و یال راست یعنی عنصر دوم کوچکتر است برپ درخت نتیجه یا خروجی را نشان میدهند.

درخت تصمیم

- برای مرتب سازی n عنصر تعداد برگ در درخت تصمیم !n است زیرا هر جایگشت از عناصر میتواند یک خروجی باشد پس ارتفاع درخت تصمیم حداقل $\lceil \log(n!) \rceil$ است.
- بنابر این در بدترین حالت برای مرتب کردن n عدد حداقل $\lceil \log(n!) \rceil$ مقایسه لازم است.
- مرتب سازی مقایسه ای در بدترین حالت از مرتبه $\Omega(nlogn)$ است و همچنین این نتیجه برای حالت متوسط نیز درست است.
- نکته : درخت دو دویی با x برگ ارتفاعش حداقل $\lceil \log(x) \rceil$ است.

درخت تصمیم

- ادغام دو لیست مرتب ۳ و ۵ عنصری درخت تصمیمش چند برگ دارد ؟

درخت تصمیم

- ادغام دو لیست مرتب ۳ و ۵ عنصری درخت تصمیمش چند برگ دارد ؟

- نتیجه : ادغام k تا لیست مرتب $\frac{n}{k}$ عنصری تعداد حالاتش در نتیجه تعداد برگ درخت تصمیم است که برابر است با $\frac{n!}{\left(\frac{n}{k}\right)!^k}$
- نتیجه : هر روشی که بخواهد k تا لیست مرتب $\frac{n}{k}$ عنصری را ادغام کند و یک لیست n تایی تحویل دهد در بدترین حال از مرتبه $\Omega\left(\log\left(\frac{n!}{\left(\frac{n}{k}\right)!^k}\right)\right) = \Omega(nlogk)$

Good luck!