# Sorting algorithms

**Selection sort, Bubble sort, Insertion sort, Shell sort, Merge sort, Heap sort, Quick sort**

Data Structures, Fall 2018
TA: Marija Stanojevic

# Selection sort

- Simplest sorting algorithm, requires $O(n^2)$ time always
- **Algorithm:** Find minimum in unsorted part of the array and puts it to the beginning of the unsorted part; then, considers that element as sorted and starts sorting from the next element

```
void sort(int arr[]) {
        int n = arr.length;
        for (int i = 0; i < n-1; i++) {
            int min_idx = i; // Find the minimum element in unsorted part of array
            for (int j = i+1; j < n; j++)
                if (arr[j] < arr[min_idx])
                    min_idx = j; // Swap the found minimum element with the first element of unsorted part
            int temp = arr[min_idx];
            arr[min_idx] = arr[i];
            arr[i] = temp;
        }
    } //time complexity: best: O(n^2), average: O(n^2), worst: O(n^2)
```

# Bubble sort

- **Idea:** easy bubbles go up in the water
- **Algorithm:** Compare every two consecutive numbers. If they are in wrong order, swap them. Move by 1. If at least one swap happen in the last pass, start new pass through the list
- Algorithm needs one **whole** pass without **any** swap to know it is sorted

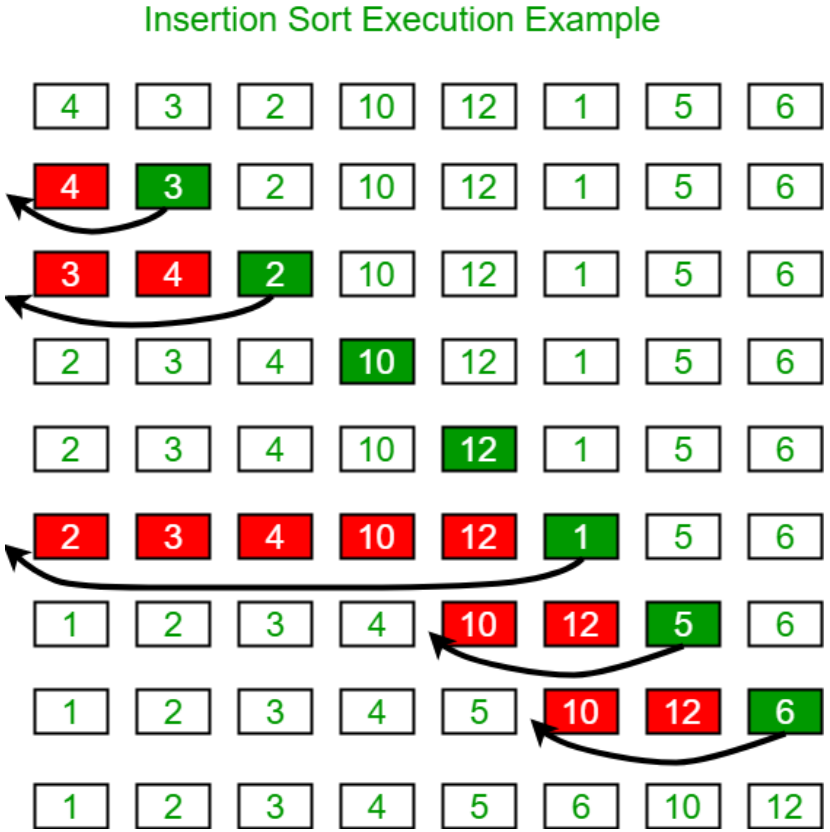
```
void sort(int arr[]) {
        int n = arr.length;
        for (int i = 0; i < n-1; i++)
            for (int j = 0; j < n-i-1; j++)
                if (arr[j] > arr[j+1]) {
                    // swap temp and arr[i]
                    int temp = arr[j];
                    arr[j] = arr[j+1];
                    arr[j+1] = temp;
                } //time complexity: best: O(n),
} // average: O(n^2), worst: O(n^2)
```

**First pass**:
( **5 1** 4 2 8 ) –> ( **1 5** 4 2 8 ), swap since 5 > 1
( 1 **5 4** 2 8 ) –> ( 1 **4 5** 2 8 ), swap since 5 > 4
( 1 4 **5 2** 8 ) –> ( 1 4 **2 5** 8 ), swap since 5 > 2
( 1 4 2 **5 8** ) –> ( 1 4 2 **5 8** ), don't swap 5 < 8
**Second Pass:**
( **1 4** 2 5 8 ) –> ( **1 4** 2 5 8 )
( 1 **4 2** 5 8 ) –> ( 1 **2 4** 5 8 ), Swap since 4 > 2
( 1 2 **4 5** 8 ) –> ( 1 2 **4 5** 8 )
( 1 2 4 **5 8** ) –> ( 1 2 4 **5 8** )
**Third Pass:**
( **1 2** 4 5 8 ) –> ( **1 2** 4 5 8 )
( 1 **2 4** 5 8 ) –> ( 1 **2 4** 5 8 )
( 1 2 **4 5** 8 ) –> ( 1 2 **4 5** 8 )
( 1 2 4 **5 8** ) –> ( 1 2 4 **5 8** )

# Insertion sort

- **Algorithm:** starting from second element compare each element with its predecessors until smaller value is found. Put current element after that value. Go to the next element.

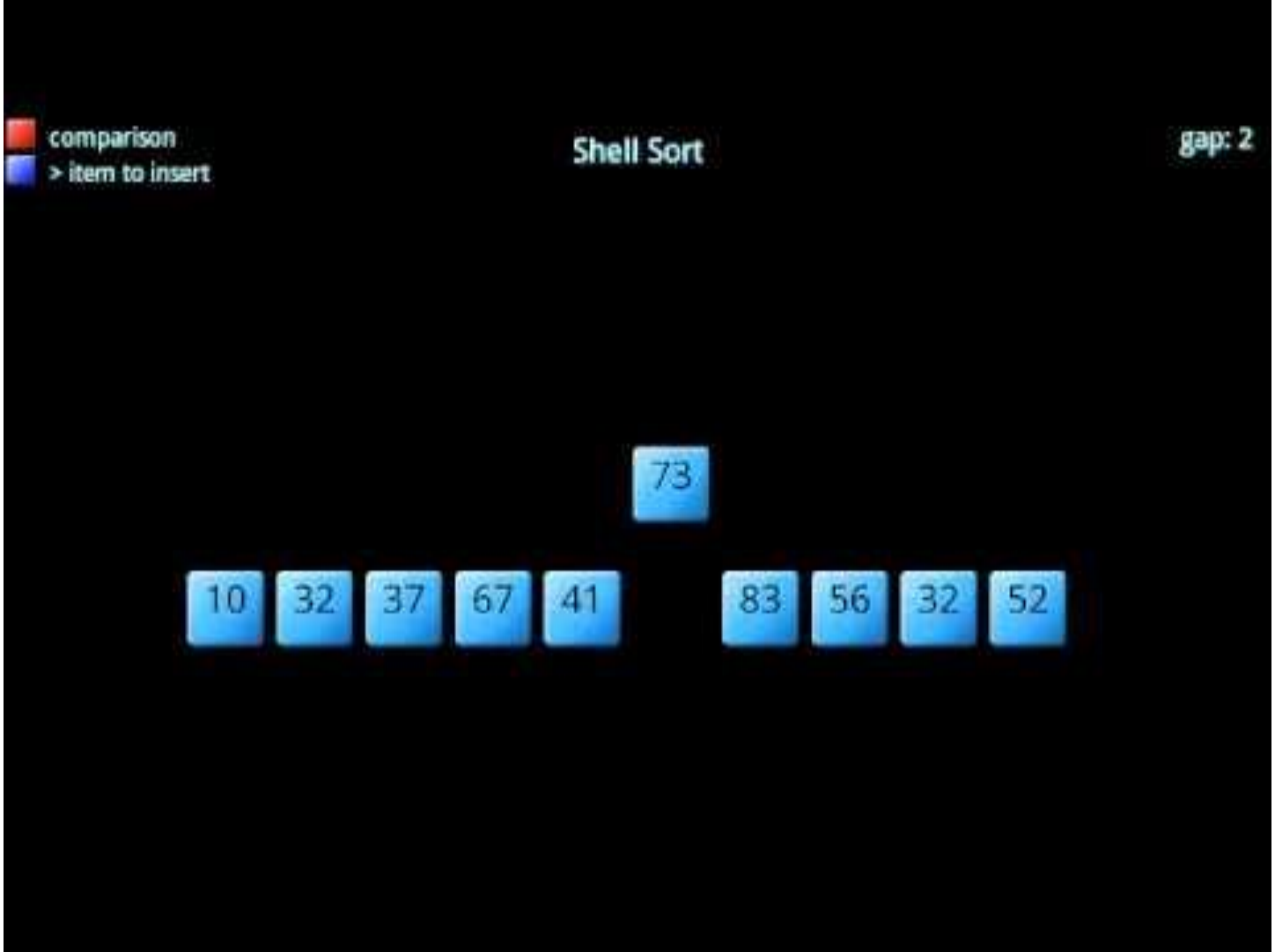
```
void sort(int arr[]) {
        int n = arr.length;
        for (int i=1; i<n; ++i) {
            int key = arr[i];
            int j = i-1;
            while (j>=0 && arr[j] > key) {
                arr[j+1] = arr[j];
                j = j-1;
            }
            arr[j+1] = key;
        } //time complexity: best: O(n),
} // average: O(n^2), worst: O(n^2)
```



# Shell sort

- **Idea**: sorting books on the shell
- **Algorithm:** Compare pairs of elements separated by X elements (gap). If they are not in good order, swap them. Divide X by 2 and repeat the process until X = 0.

```
void sort(int arr[]) {
        int n = arr.length;
        for (int gap = n/2; gap > 0; gap /= 2)
            for (int i = gap; i < n; i += 1) {
                int temp = arr[i];
                int j;
                for (j = i; j >= gap && arr[j - gap] > temp; j -= gap)
                    arr[j] = arr[j - gap];
                arr[j] = temp;
            }
} //time complexity: best: O(n log n), worst: O(n^(4/3))
```


comparison
> item to insert
Shell Sort
gap: 2
73
10
32
37
67
41
83
56
32
52

# Merge sort

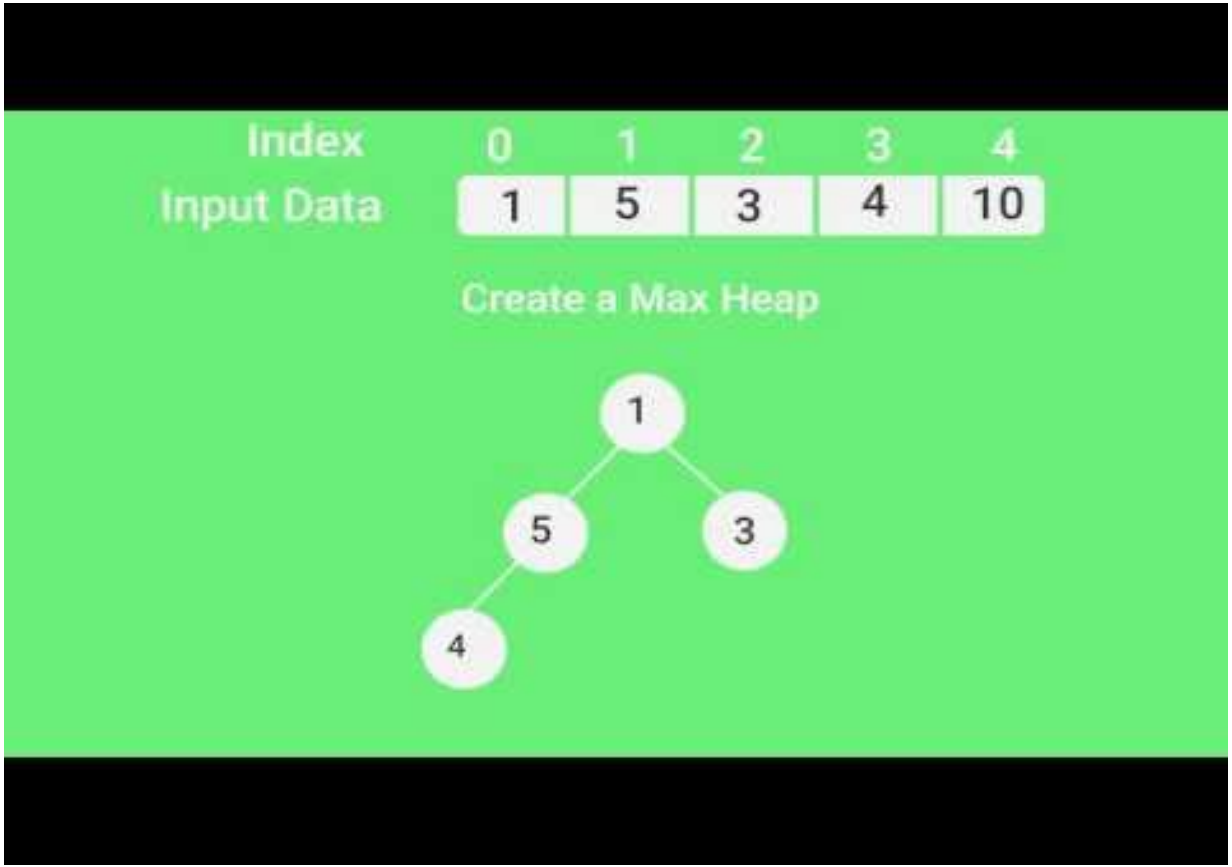
- Highly parallelizable => divide on n machines => time complexity O(log n)
- **Algorithm:** Divide array into halfs recursively and once we array has only two order those. Then return back into bigger case and merge those sorted arrays.
- **Space complexity**: O(n);

```
void mergeSort(int[] a, int n) {
  if (n < 2) return;
  int mid = n / 2;
  int[] l = new int[mid];
  int[] r = new int[n - mid];
  for (int i = 0; i < mid; i++) {
    l[i] = a[i];
  }
  for (int i = mid; i < n; i++) {
    r[i - mid] = a[i];
  }
  mergeSort(l, mid);
  mergeSort(r, n - mid);
  merge(a, l, r, mid, n - mid);
} // Time complexity: best:
O(n log n), average: O(n log n),
worst: O(n log n)
```

```
void merge(int[] a, int[] l,
int[] r, int left, int right) {
  int i = 0, j = 0, k = 0;
  while (i<left && j < right) {
    if (l[i] < r[j]) {
      a[k++] = l[i++];
    } else {
      a[k++] = r[j++];
    }
  }
  while (i < left) {
    a[k++] = l[i++];
  }
  while (j < right) {
    a[k++] = r[j++];
  }
}
```


Merge Sort
HackerRank

# Heap sort




```
void sort(int arr[]) {
  int n = arr.length;
  for (int i=n/2-1; i >= 0; i--)
    heap(arr, n, i);
  for (int i=n-1; i>=0; i--) {
    int temp = arr[0];
    arr[0] = arr[i];
    arr[i] = temp;
    heap(arr, i, 0);
  }
}
```

```
void heap(int arr[],int n,int i) {
  int largest = i;
  int l = 2*i + 1;
  int r = 2*i + 2;
  if (l<n && arr[l]>arr[largest])
    largest = l;
  if (r<n && arr[r]>arr[largest])
    largest = r;
  if (largest != i) {
    int swap = arr[i];
    arr[i] = arr[largest];
    arr[largest] = swap;
    heapify(arr, n, largest);
  } // Time: best: O(n log n), avg:
} // O(n log n), worst: O(n log n)
```

# Quick sort

- Similar to MergeSort; both are used in Java for sorting
- **Algorithm:** Picks an element as pivot and partitions around it. Partitioning exchange elements so that elements smaller than pivot go to the left and elements bigger than pivot go to the right. Then, sort is called for each of those two parts recursively.
- **Pivot** can be chosen in different ways:
  - Pick first / last element
  - Pick a random element
  - Pick median

```
void sort(int arr[], int low, int high) {
  if (low < high) {
    int pi = part(arr, low, high);
     sort(arr, low, pi-1);
     sort(arr, pi+1, high);
  } // Time complexity: best: O(n),
} // average: O(n log n), worst: O(n^2)
```

```
int part(int arr[],int low,int high) {
  int pivot = arr[high];
  int i = (low-1);
  for (int j=low; j<high; j++) {
    if (arr[j] <= pivot) {
      i++;
      int temp = arr[i];
      arr[i] = arr[j];
      arr[j] = temp;
    }
  }
  int temp = arr[i+1];
  arr[i+1] = arr[high];
  arr[high] = temp;
  return i+1;
}
```


Quicksort
HackerRank