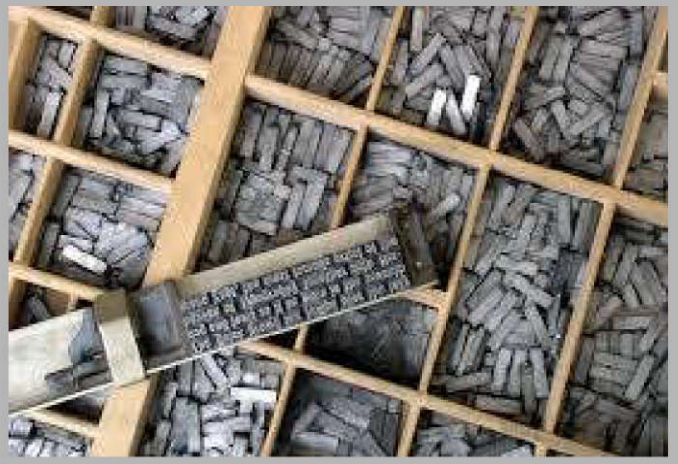
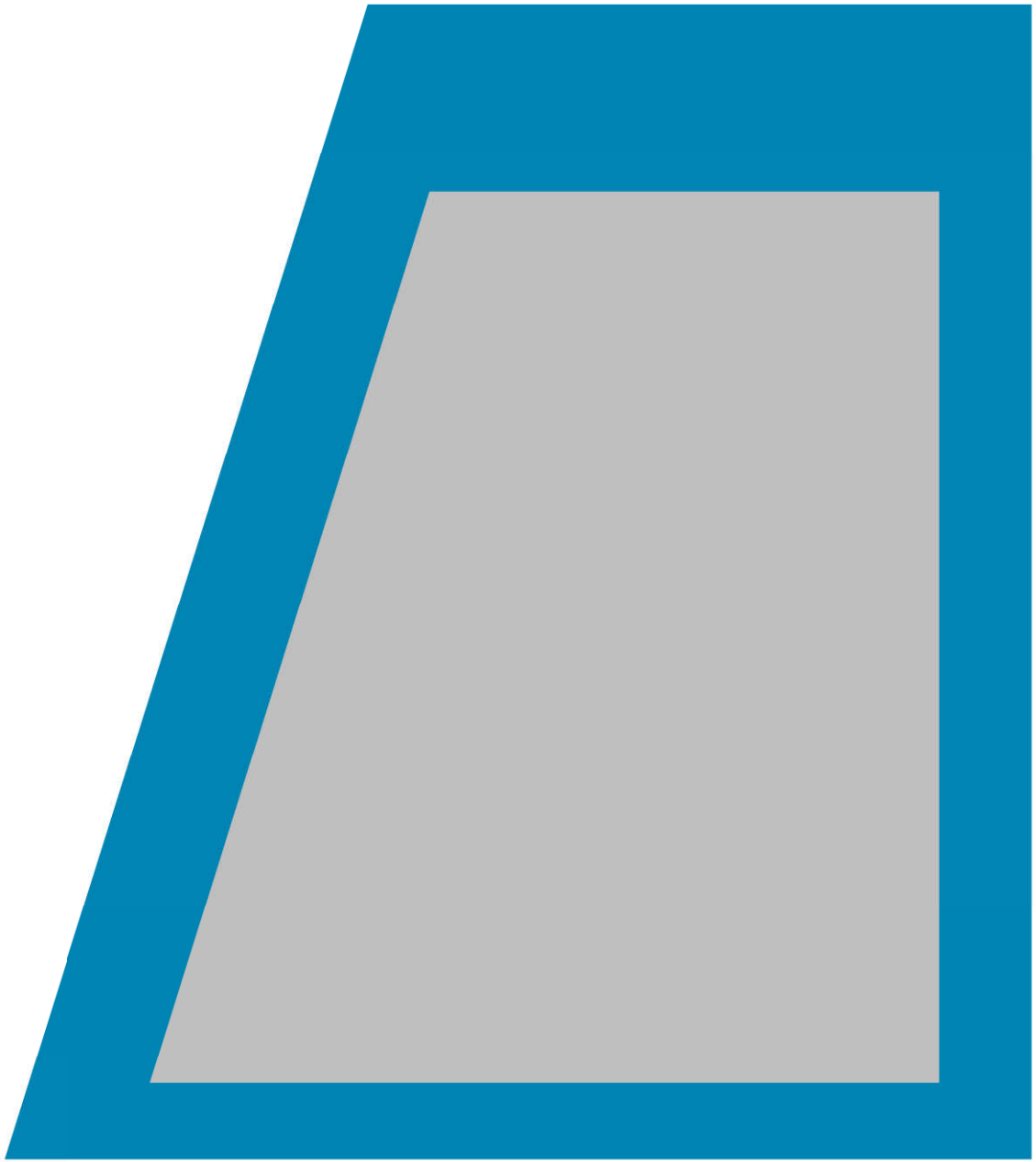
ITD62-124 Data Structure

Sorting Algorithms



Outline:

- Insertion sort
- Bubble sort
- Heap sort
- Shell sort
- Quick sort
- Merge sort
- Radix sort



Introduction

- Input:

  A sequence of n numbers $a_1, a_2, \ldots, a_n$

- Output:

  A permutation (reordering) $a_1', a_2', \ldots, a_n'$ of the input sequence such that $a_1' \leq a_2' \leq \cdots \leq a_n'$

Introduction

- **Importance of sorting algorithms**

  There are a variety of situations that we can encounter

  ✓have randomly ordered keys

  ✓all keys distinct

  ✓How large is the set of keys to be ordered?

  ✓Need guaranteed performance?

Insertion Sort

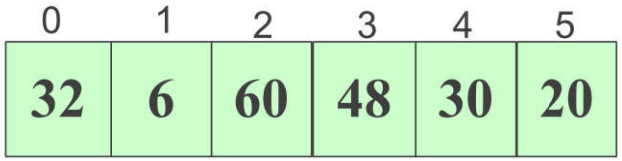
- insert each item into its proper place in the final array
- moving the current data elements past the already sorted values and repeatedly interchanging it with the preceding value until it is in its correct place
- the array of values to be sorted is divided into two sets (one that stores sorted values and another that contains unsorted values)

Insertion Sort

- Initially, data[0] is the only element in the sorted set
- In Pass 1, data[1] will be placed either before or after data[0], therefore the array is sorted
- In Pass 2, data[2] will be placed either before data[0], in between data[0] and data[1], or after data[1]
- In Pass 3, data[3] will be placed in its proper place
- In Pass n-1, data[n-1] will be placed in its proper place to keep the array sorted

Insertion Sort

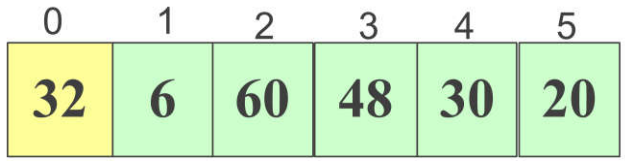
- Example: sort a given array by insertion sort



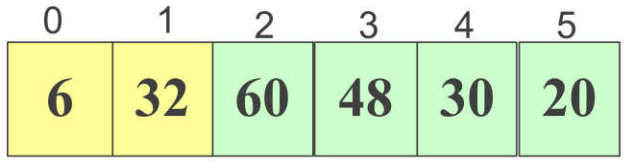
Insertion Sort

- Solution:

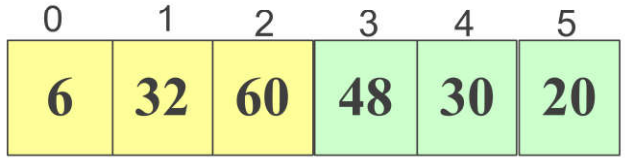
✓Initially:



✓Pass 1:



✓Pass 2:



Insertion Sort

- Solution:

✓Pass 3:

0	1	2	3	4	5
6	32	48	60	30	20

✓Pass 4:

0	1	2	3	4	5
6	30	32	48	60	20

✓Pass 5:

0	1	2	3	4	5
6	20	30	32	48	60

Bubble Sort

✓Consecutive adjacent pairs of elements in the array are compared with each other

✓Two adjacent elements are interchanged if they are found to be out of order with respect to each other

✓This process will continue till the list of unsorted elements exhausts

Bubble Sort

✓In Pass 1, data[0] and data[1] are compared, then data[1] is compared with data[2], data[2] is compared with data[3], and so on. Finally, data[n-2] is compared with data[n-1]. The biggest element is placed at the highest index of the array.

✓In Pass 2, data[0] and data[1] are compared, then data[1] is compared with data[2], data[2] is compared with data[3], and so on. Finally, data[n-3] is compared with data[n-2]. The second biggest element is placed at the second highest index of the array.

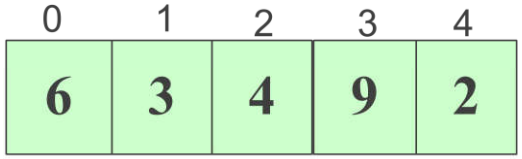
Bubble Sort

✓In Pass 3, data[0] and data[1] are compared, then data[1] is compared with data[2], data[2] is compared with data[3], and so on. Finally, data[n-4] is compared with data[n-3]. The third biggest element is placed at the third highest index of the array.

✓In Pass n-1, data[0] and data[1] are compared so that data[0] < data[1]. After this step, all the elements of the array are arranged in ascending order.

Bubble Sort

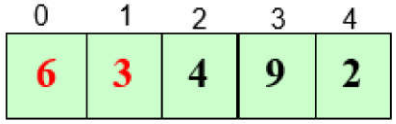
- Example: sort a given array by bubble sort



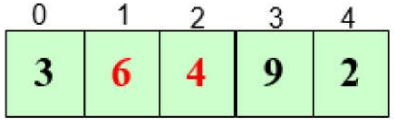
Bubble Sort

- Solution: Pass 1

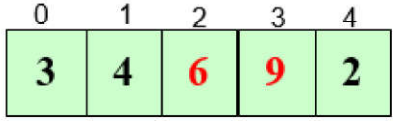
✓ Step 1:



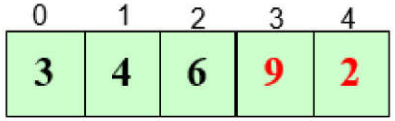
✓ Step 2:

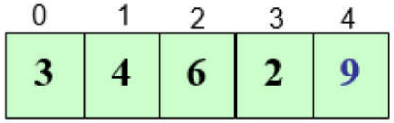


✓ Step 3:



✓ Step 4:

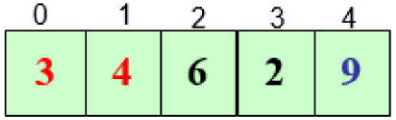




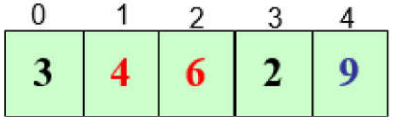
Bubble Sort

- Solution: Pass 2

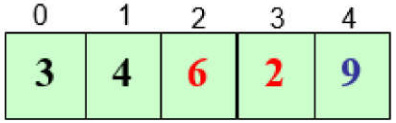
✓ Step 1:



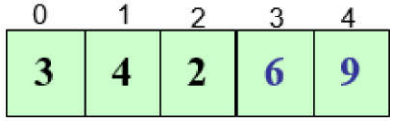
✓ Step 2:



✓ Step 3:



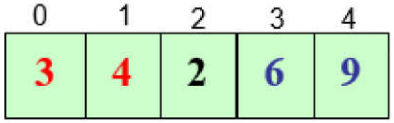
✓ Step 4:



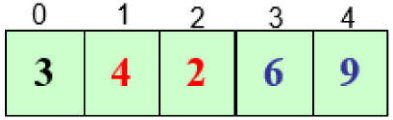
Bubble Sort

- Solution: Pass 3

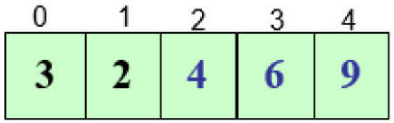
✓ Step 1:



✓ Step 2:



✓ Step 3:



Bubble Sort

- Solution: Pass 4

✓ Step 1:

0	1	2	3	4
3	2	4	6	9

✓ Step 2:

0	1	2	3	4
2	3	4	6	9

Heap Sort

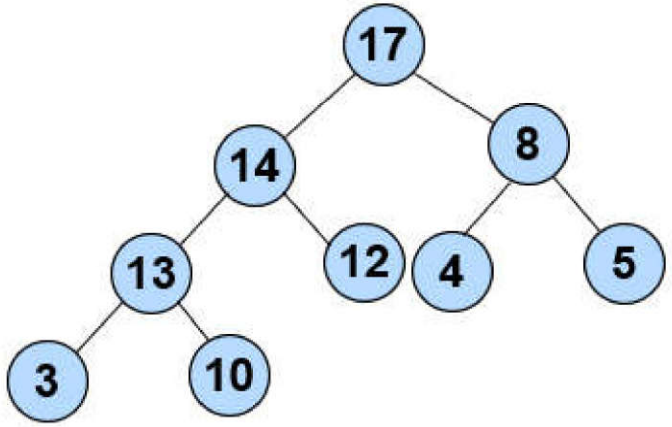
- ✓Invented by John Williams
- ✓To sort an array in ascending order, heap sort puts the largest elements at the end of the array, then the second largest in front of it, and so on
- ✓Start from the end of the array by finding the largest elements
- ✓Heap sort uses a heap

Heap Sort

- Example: sort the heap using heap sort

  17 14 8 13 12 4 5 3 10

Heap:

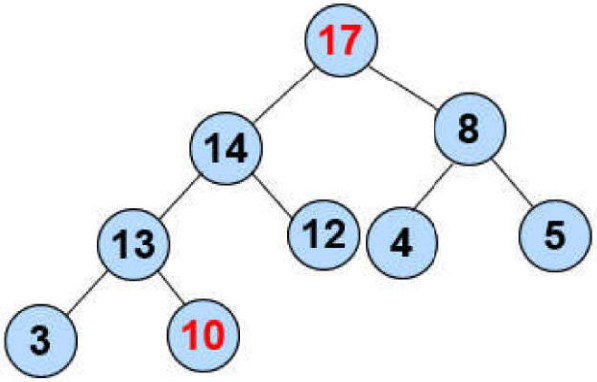


Heap Sort

- Solution:

17 14 8 13 12 4 5 3 10

Heap:

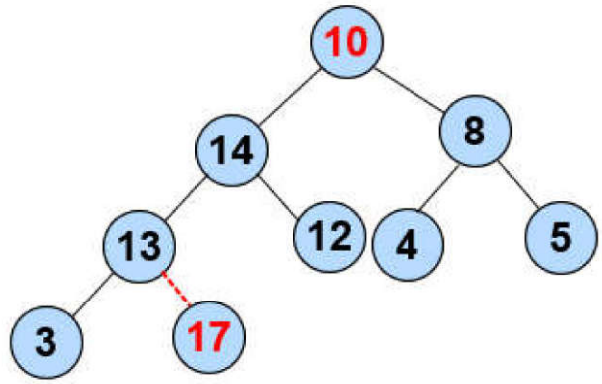


Heap Sort

- Solution:

10 14 8 13 12 4 5 3 17

Heap:

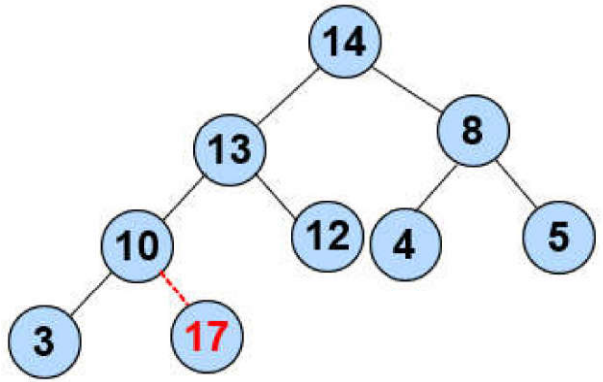


Heap Sort

- Solution:

14 13 8 10 12 4 5 3 17

Heap:

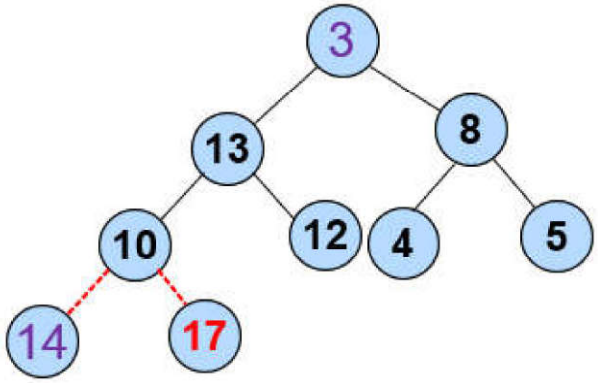


Heap Sort

- Solution:

14 13 8 10 12 4 5 3 17

Heap:

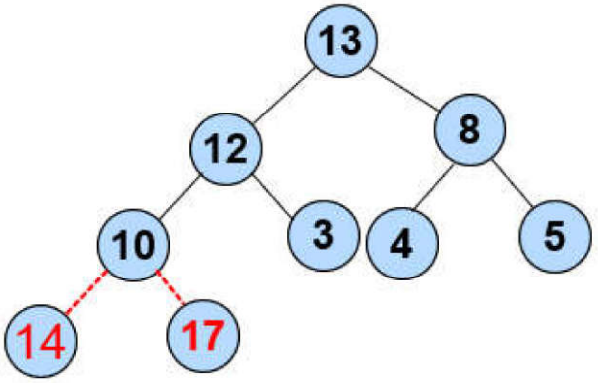


Heap Sort

- Solution:

13 12 8 10 3 4 5 14 17

Heap:

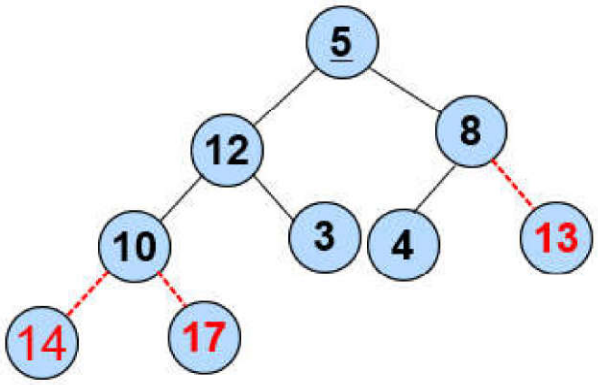


Heap Sort

- Solution:

13 12 8 10 3 4 5 14 17

Heap:

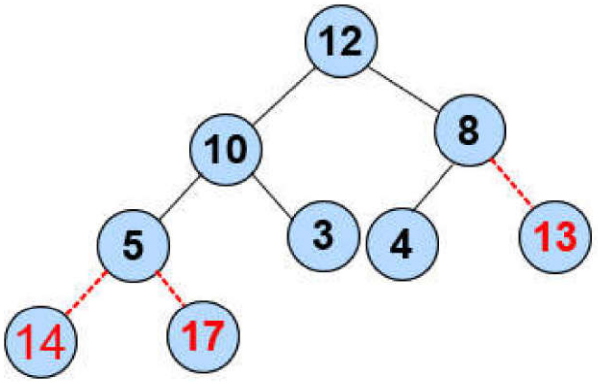


Heap Sort

- Solution:

12 10 8 5 3 4 13 14 17

Heap:

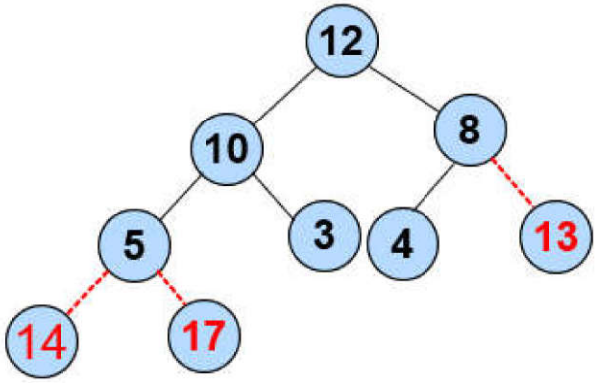


Heap Sort

- Solution:

12 10 8 5 3 4 13 14 17

Heap:

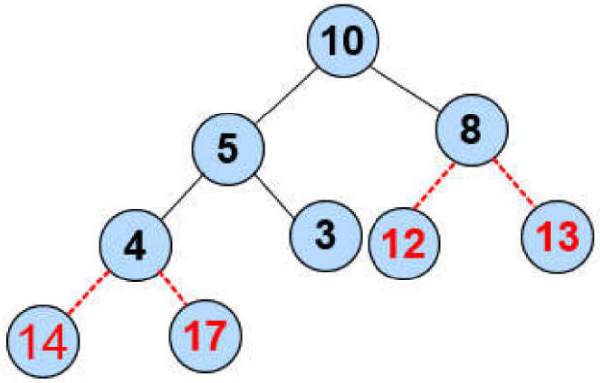


Heap Sort

- Solution:

12 10 8 5 3 4 13 14 17

Heap:

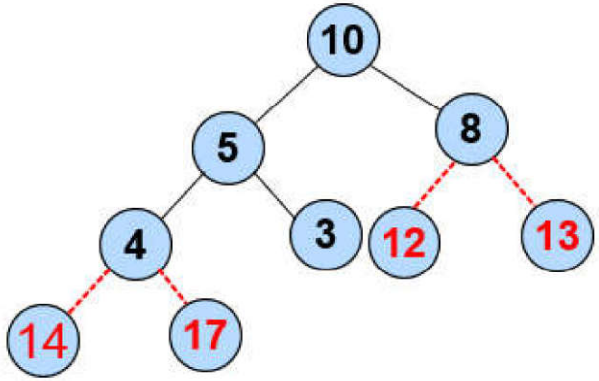


Heap Sort

- Solution:

10 5 8 4 3 12 13 14 17

Heap:

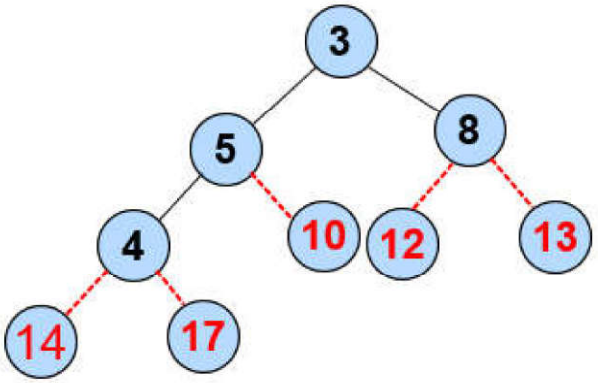


Heap Sort

- Solution:

10 5 8 4 3 12 13 14 17

Heap:

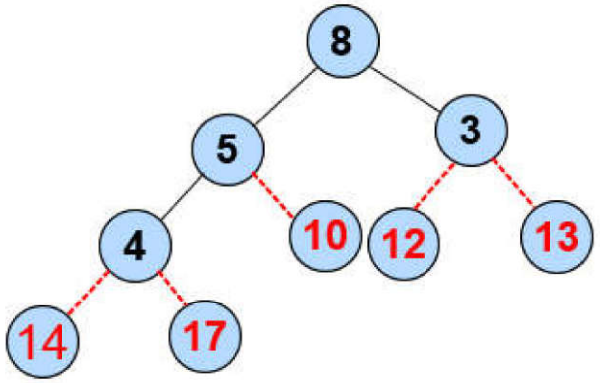


Heap Sort

- Solution:

8 5 3 4 10 12 13 14 17

Heap:

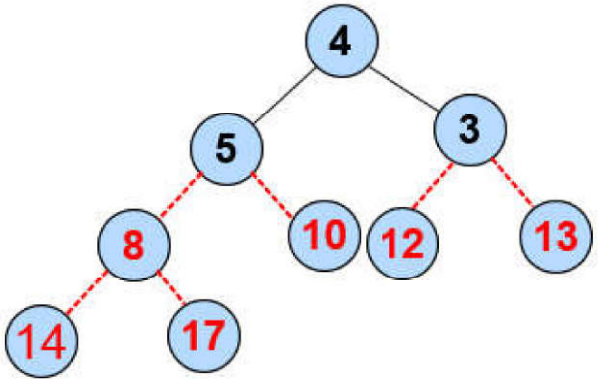


Heap Sort

- Solution:

8 5 3 4 10 12 13 14 17

Heap:

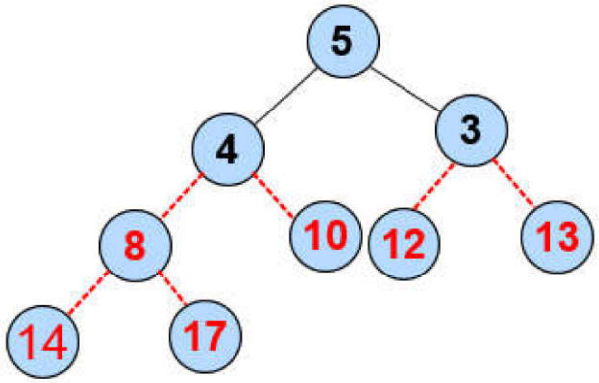


Heap Sort

- Solution:

5 4 3 8 10 12 13 14 17

Heap:

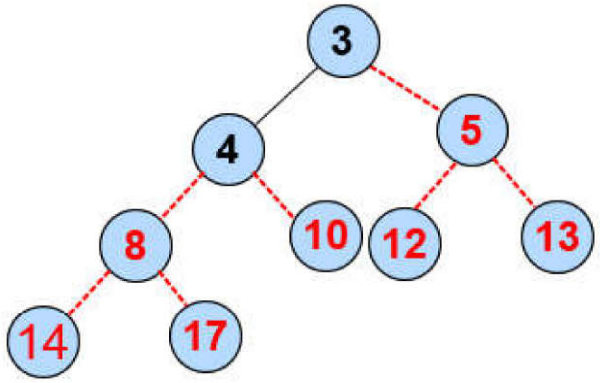


Heap Sort

- Solution:

5 4 3 8 10 12 13 14 17

Heap:

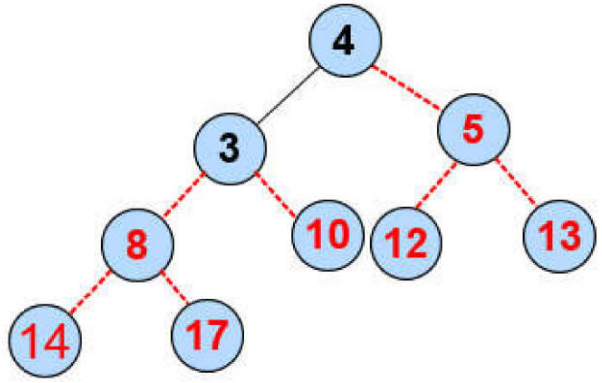


Heap Sort

- Solution:

4 3 5 8 10 12 13 14 17

Heap:

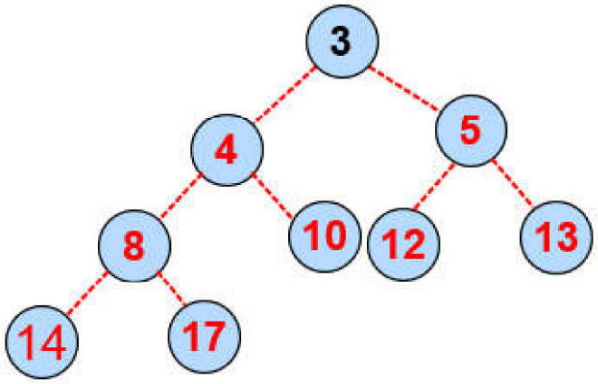


Heap Sort

- Solution:

4 3 5 8 10 12 13 14 17

Heap:

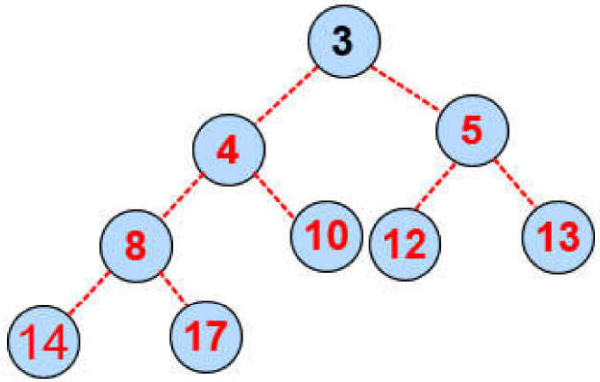


Heap Sort

- Solution:

3 4 5 8 10 12 13 14 17

Heap:



Shell Sort

- ✓Invented by Donald Shell in 1959
- ✓Division of an array data into several subarrays
- ✓Elements spaced farther apart are compared first, then the elements closer to each other are compared, …, until adjacent elements are compared on the last pass
- ✓Shell sort improves on the efficiency of insertion sort by quickly shifting values to their destination
- ✓elements are sorted in multiple passes and in each pass, data are taken with smaller and smaller gap sizes
- ✓The last step of Shell sort is a plain insertion sort

Shell Sort

```
Shell_Sort(Arr, N)

Step 1: SET FLAG = 1, GAP_SIZE = N
Step 2: Repeat Steps 3 to 6 while FLAG = 1 OR GAP_SIZE > 1
Step 3:          SET FLAG = 0
Step 4:          SET GAP_SIZE = (GAP_SIZE + 1) / 2
Step 5:          Repeat Step 6 for I = 0 to I < (N - GAP_SIZE)
Step 6:                   IF Arr[I + GAP_SIZE] > Arr[I]
                                  SWAP Arr[I + GAP_SIZE], Arr[I]
                                  SET FLAG = 0
Step 7: END
```

Shell Sort

- Example: sort a given array by shell sort

63, 19, 7, 90, 81, 36, 54, 45, 72, 27, 22, 9, 41, 59, 33

Shell Sort

- Solution:
  - ✓Step 1:

63	19	7	90	81	36	54	45
72	27	22	9	41	59	33	

63	19	7	9	41	36	33	45
72	27	22	90	81	59	54	

63, 19, 7, 9, 41, 36, 33, 45, 72, 27, 22, 90, 81, 59, 54

Shell Sort

✓Step 2:

63	19	7	9
41	36	33	45
72	27	22	90
81	59	54	

41	19	7	9
63	27	22	45
72	36	33	90
81	59	54	

41, 19, 7, 9, 63, 27, 22, 45, 72, 36, 33, 90, 81, 59, 54

Shell Sort

✓Step 3:

41	19
7	9
63	27
22	45
72	36
33	90
81	59
54	

7	9
22	19
33	27
41	36
54	45
63	59
72	90
81	

7, 9, 22, 19, 33, 27, 41, 36, 54, 45, 63, 59, 72, 90, 81

Shell Sort

✓Step 4:

7	7
9	9
22	19
19	22
33	27
27	33
41	36
36	41
54	45
45	54
63	59
59	63
72	72
90	81
81	90

Quick Sort

- **Quick sort**
  - ✓ The original array is divided into two subarrays
    - the first array contains elements less than or equal to a chosen key called the pivot
    - the second array contains elements equal to or greater than the pivot
  - ✓ The two subarrays can be sorted separately

Quick Sort

- Example: sort a given array by quick sort

  27 15 22 37 11 59 18 50 42

Quick Sort

- **Solution**:

27 15 22 37 11 59 18 50 42

27 15 22 37 11 59 18 50 42

18 15 22 37 11 59 27 50 42

18 15 22 37 11 59 27 50 42

18 15 22 37 11 59 27 50 42

Quick Sort

- **Solution**:

18 15 22 37 11 59 27 50 42
18 15 22 27 11 59 37 50 42
18 15 22 27 11 59 37 50 42
18 15 22 27 11 59 37 50 42
18 15 22 11 27 59 37 50 42
18 15 22 11 27 59 37 50 42

Pivot = 27, first subarray = 18 15 22 11, second subarray = 59 37 50 42

Quick Sort

- Solution:

18 15 22 11 27 59 37 50 42

- Sort the first subarray

18 15 22 11

11 15 22 18

11 15 22 18

11 15 22 18

11 15 18 22

11 15 18 22

Pivot = 18, first subarray = 11 15, second subarray = 22

Quick Sort

- Solution:

11 15 18 22 27 59 37 50 42

- Sort the first subarray

11 15

11 15

11 15 18 22 27 59 37 50 42

Quick Sort

- Solution:

11 15 18 22 27 59 37 50 42

- Sort the second subarray(59 37 50 42)

59 37 50 42

42 37 50 59

42 37 50 59

42 37 50 59

42 37 50 59

Pivot = 59, first subarray = 42 37 50

Quick Sort

- Solution:

11 15 18 22 27 42 37 50 59

- Sort the subarray (42 37 50)

42 37 50

42 37 50

37 42 50

37 42 50

Pivot = 42

11 15 18 22 27 37 42 50 59

Merge Sort

- **Merge sort**
  - ✓ Merging sorted halves of an array into one sorted array

Merge Sort

- **Example**: sort a given array by merge sort

3 10 8 6 12 7 5 4 24

✓Solution:

3 10 8 6 12 7 5 4 24

3 10 8 6 12 7 5 4 24

3 10 8 6 1 2 7 5 4 24

3 10 8 6 1 2 7 5 4 24

3 10 8 6 1 2 7 5 4 24

Merge Sort

3 8 10 6 1 2 5 7 4 24

3 6 8 10 12 4 5 7 24

3 4 5 6 7 8 10 12 24

Radix Sort

- **Radix sort**

  ✓ sorts data with integer keys by grouping keys by the individual digits which share the same significant position and value

Radix Sort

- **Example**: sort a given array by radix sort

  12 1234 9 7234 67 9191 733 197 7 3

Radix Sort

- **Solution**: 12 1234 9 7234 67 9191 733 197 7 3

Step 1:

bucket 0:
bucket 1: 9191
bucket 2: 12
bucket 3: 733 3
bucket 4: 1234 7234
bucket 5:
bucket 6:
bucket 7: 67 197 7
bucket 8:
bucket 9: 9

9191 12 733 3 1234 7234 67 197 7 9

Radix Sort

- **Solution**: 9191 12 733 3 1234 7234 67 197 7 9

  Step 2:

  bucket 0: 3 7 9
  bucket 1: 12
  bucket 2:
  bucket 3: 733 1234 7234
  bucket 4:
  bucket 5:
  bucket 6: 67
  bucket 7:
  bucket 8:
  bucket 9: 9191 197

  3 7 9 12 733 1234 7234 67 9191 197

Radix Sort

- **Solution**: 3 7 9 12 733 1234 7234 67 9191 197

  Step 3:

  bucket 0: 3 7 9 12 67
  bucket 1: 9191 197
  bucket 2: 1234 7234
  bucket 3:
  bucket 4:
  bucket 5:
  bucket 6:
  bucket 7: 733
  bucket 8:
  bucket 9:

  3 7 9 12 67 9191 197 1234 7234 733

Radix Sort

- **Solution**: 3 7 9 12 67 9191 197 1234 7234 733

  Step 4:

  bucket 0: 3 7 9 12 67 197 733
  bucket 1: 1234
  bucket 2:
  bucket 3:
  bucket 4:
  bucket 5:
  bucket 6:
  bucket 7: 7234
  bucket 8:
  bucket 9: 9191

3 7 9 12 67 197 733 1234 7234 9191

Class Activity

Q & A

Formative Assessment