GIT Department of Computer Engineering

CSE 222/505 – Spring 2020

Homework 6 – Question 1

Shell Sort

A =

1	2	3	4	5	6	7	8	9	10

Gap value = 5, now we create our subarrays for this gap value:

Subarray1:

1	6

Subarray2:

2	7

Subarray3:

3	8

Subarray4:

4	9

Subarray5:

5	10

- Sorting subarrays:

Subarray1: 1<6 correct. 1 comparison, no swap.

Subarray2: 2<7 correct. 1 comparison, no swap.

Subarray3: 3<8 correct. 1 comparison, no swap.

Subarray4: 4<9 correct. 1 comparison, no swap.

Subarray5: 5<10 correct. 1 comparison, no swap.

Array is :

1	2	3	4	5	6	7	8	9	10

Then our gap value will change, 5/2=2 is new gap value. Finding subarrays with gap value 2:

Subarray1 :

1	3	5	7	9

Subarray2:

2	4	6	8	10

- Sorting subarrray1: 4 comparison (1<3, 3<5 ,5<7, 7<9), no swap.
- Sorting subarray2: 4 comparison (2<4, 4<6, 6<8, 8<10), no swap

After that array is :

1	2	3	4	5	6	7	8	9	10

Finally , 2/2=1, 1 is new gap value and we sort the array again with gap value 1:

Subarray1:

1	2	3	4	5	6	7	8	9	10

We check as an insertion sort while gap is 1.

9 comparison made, and array is sorted. No swap.

In total, 22 comparison made and no swap. Array is sorted.

A =

1	2	3	4	5	6	7	8	9	10

B =

10	9	8	7	6	5	4	3	2	1

Gap value n/2=10/2=5. We divide array to subarrays and sort them:

Subarray1:

10	5

Subarray2:

9	4

Subarray3:

8	3

Subarray4:

7	2

Subarray5:

6	1

- Sorting subarrays:

Subarray1: 10>5, 1 comparison, 10-5 swapped,1 swap.

Subarray2: 9>4, 1 comparison, 9-4 swapped,1swap.

Subarray3: 8>3, 1 comparison, 8-3 swapped,1 swap.

Subarray4: 7>2,1 comparison, 7-2 swapped,1 swap.

Subarray5: 6>1, 1 comparison, 6-1 swaped,1 swap.

Current array after swaps:

5	4	3	2	1	10	9	8	7	6

New gap value: 5/2 = 2. We divide subarrays with gap value 2.

Subarray1:

5	3	1	9	7

Subarray2:

4	2	10	8	6

- Sort subarray1:

Initial subarray1,choose first element as sorted array:

5	3	1	9	7

Start from index 1, 3<5 , swap 3-5, 1 comparison,1 swap.

3	5	1	9	7

1<5, swap 1-5, 1 comparison,1 swap.

3	1	5	9	7

1<3, swap 1-3, 1 comparison,1 swap.

1	3	5	9	7

9>5, but 9<7, swap 9-7, 2 comparison,1swap.

1	3	5	7	9

Subarray1 is sorted. 5 comparison,4 swap made.

1	3	5	7	9

- Sort subarray2:

Initial subarray2,choose first element as sorted array:

4	2	10	8	6

Start from index 1, 2<4 , swap 2-4, 1 comparison,1 swap.

2	4	10	8	6

10>8, swap 10-8 , 1 comparison 1 swap.

2	4	8	10	6

8>4, 10>8 but 6<10 swap 10-6, 3 comparison,1 swap.

2	4	8	6	10

6<8, swap 6-8, 1 comparison,1 swap.

2	4	6	8	10

Subarray2 is sorted. 6 comparison,4 swap made.

2	4	6	8	10

Now put subarray1 and subarray2, current array is:

1	2	3	4	5	6	7	8	9	10

Our new gap is 2/2=1. Subarray with gap value 1:

1	2	3	4	5	6	7	8	9	10

We make insertion sort after gap 1 and there is 9 comparison made, no swap,array is sorted.

Totally 16 comparison and 13 swap made.

B =

1	2	3	4	5	6	7	8	9	10

C =

5	2	13	9	1	7	6	8	1	15	4	11

Lets take first gap value is 5 and divide it into subarrays according to gap value 5:

Subarray1:

5	7	4

Subarray2:

2	6	11

Subarray3:

13	8

Subarray4:

9	1

Subarray5:

1	15

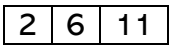
Sorting subarrays:

- Sort subarray1:

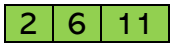
5<7, 7>4 swap 7-4 , 2 comparison,1 swap

5>4 , swap 5-4 , 1 comparison,1 swap

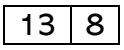
- Sort subarray2:



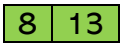
2<6, 6<11, 2 comparison, no swap



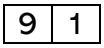
- Sort subarray3:



13>8, swap 13-8, 1 comparison,1 swap

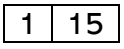


- Sort subarray4:

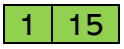


9>1, swap 9-1, 1 comparison,1 swap

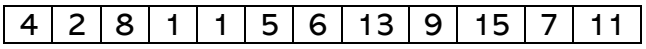
- Sort subarray5:



1<15, 1 comparison, no swap



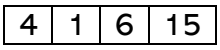
Current array after sorting subarrays with gap 5:



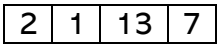
Lets say new gap value is =3.

Subarrays when gap is 3:

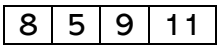
Subarray1:



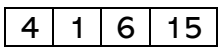
Subarray2:



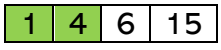
Subarray3:



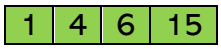
- Sort subarray1:



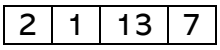
4>1, swap 4-1, 6>4,15>6, 3 comparison,1 swap.



Subarray1 is sorted now.

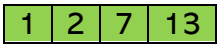


- Sort subarray2:



2>1 swap 2-1, 1 comparison,1 swap.

2<13, 13>7, swap 13-7, 2 comparison 1 swap .



2<7, subarrays is sorted. 1 comparison.

- Sort subarray3:

8	5	9	11

5<8, swap 5-8. 1 comparison,1 swap.

5	8	9	11

8<9, 9<11, 2 comparison, subarrays is sorted.

5	8	9	11

After sorting subarrays with gap 3, current array is:

1	1	5	4	2	8	6	7	9	15	13	11

Now we take gap value as 1.

- Sort subarray:

1 <= 1, 1<5, but 5>4, swap 4-5, 3 comparison 1 swap.

1	1	4	5	2	8	6	7	9	15	13	11

2<5, swap 2-5 ,1 comparison,1 swap.

1	1	4	2	5	8	6	7	9	15	13	11

2<4,swap 2-4, 1 comparison,1 swap.

1	1	2	4	5	8	6	7	9	15	13	11

5>4, 8>5, 6<8, swap 6-8, 3 comparison,1swap.

1	1	2	4	5	6	8	7	9	15	13	11

7<8, swap 7-8,1 comparison,1 swap.

1	1	2	4	5	6	7	8	9	15	13	11

8<9, 9<15,15>13, swap 15-13 3 comparison,1 swap.

1	1	2	4	5	6	7	8	9	13	15	11

15>11 swap 15-11,1 comparison, 1 swap.

1	1	2	4	5	6	7	8	9	13	11	15

11<13, swap 11-13, 1 comparison, 1 swap.

1	1	2	4	5	6	7	8	9	11	13	15

Array is sorted after 32 comparisons and 16 swap.

C =

1	1	2	4	5	6	7	8	9	11	13	15

D =

S	B	I	M	H	Q	C	L	R	E	P	K

Lets take gap value as 5, divide array to subarrays.

Subarray1:

S	Q	P

Subarray2:

B	C	K

Subarray3:

I	L

Subarray4:

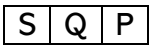
M	R

Subarray5:

H	E

Sorting subarrays;

- Sort subarray1:



Q<S, swap Q-S, 1 comparison 1 swap.

S>P, swap S-P, 1 comparison 1 swap.

P<Q, swap P-Q, 1 comparison 1 swap.

- Sort subarray2:

B<C, C<K ,2 comparisons no swap.

- Sort subarray3:

I<L, 1 comparison no swap.

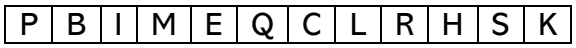
- Sort subarray4:

M<R, 1 comparison no swap.

- Sort subarray5:

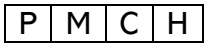
E<H, swap E-H, 1 comparison 1 swap.

After sorting with gap 5 our current array is:



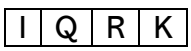
Now, new gap value is 3, we divide our current array to subarrays using gap 3:

Subarray1:

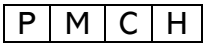


Subarray2:

Subarray3:



- Sort subarray1:



M<P, swap P-M, 1 comparison,1 swap.

C<P ,swap C-P and C<M swap C-M ,

2 comparison,2 swap.

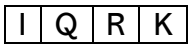
H<P, swap P-H, and H<M swap M-H,

2 comparison,2 swap.

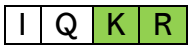
- Sort subarray2:

E>B, L>E, S>L, 3 comparisons, no swap.

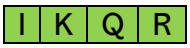
- Sort subarray3:



I<Q, Q<R, R>K, swap R-K, 3 comparisons 1 swap



K<Q, swap K-Q 1comparison 1 swap.



After sorting with gap 3 , current array is:

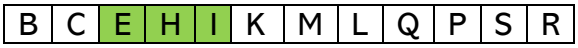
Now take gap as 1.

- Sort subarray1:

B<C, swap B-C, 1 comparison, 1 swap.

I>H, swap I-H, C<H, 2 comparison,1 swap.

E<I, swap E-I, E<H swap E-H, 2 comparison,



2 swap.

K<M, M>L, swap M-L, 2 comparisons,1 swap.

Q>P, swap P-Q, 1 comparison,1 swap.

Q<S, R<S, swap S-R, 2 comparison,1 swap.

Array is sorted after 30 comparisons and 18 swaps.

D =

Merge Sort

A = | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

Step1: | 1 | 2 | 3 | 4 | 5 | | 6 | 7 | 8 | 9 | 10 |

Step2: | 1 | 2 | 3 | | 4 | 5 | | 6 | 7 | 8 | | 9 | 10 |

Step3: | 1 | 2 | | 3 | | 4 | | 5 | | 6 | 7 | | 8 | | 9 | | 10 |

Step4: | 1 | | 2 | | 3 | | 4 | | 5 | | 6 | | 7 | | 8 | | 9 | | 10 |

We divide our array until all subarrays are at size1, now we will merge them.

Step5: | 1 | 2 | | 3 | | 4 | 5 | | 6 | 7 | | 8 | | 9 | 10 |

1<2, 3 is odd, 4<5 and 6<7, 8 is odd, 9<10. 4 comparison made, no swap

Step6: | 1 | 2 | 3 | | 4 | 5 | | 6 | 7 | 8 | | 9 | 10 |

1<3 and 2<3, 3 merged, 6<8 and 7<8, 8 is merged. 4 comparison made, no swap

Step7: | 1 | 2 | 3 | 4 | 5 | | 6 | 7 | 8 | 9 | 10 |

2 comparison, no swap.

Step8: | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

1 comparison made.

In total we divide and merge our array with 11 comparisons and no swap.

A= | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

B = | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |

Step1: | 10 | 9 | 8 | 7 | 6 | | 5 | 4 | 3 | 2 | 1 |

Step2: | 10 | 9 | 8 | | 7 | 6 | | 5 | 4 | 3 | | 2 | 1 |

Step3 | 10 | 9 | | 8 | | 7 | | 6 | | 5 | 4 | | 3 | | 2 | | 1 |

Step4: | 10 | | 9 | | 8 | | 7 | | 6 | | 5 | | 4 | | 3 | | 2 | | 1 |

We divide our array until all subarrays are at size1, now we will merge them.

Step5: | 9 | 10 | | 8 | | 6 | 7 | | 4 | 5 | | 3 | | 1 | 2 |

9<10 merge them,6<7 merged, 4<5 and 1<2 merged.4 comparisons made

Step6: | 8 | 9 | 10 | | 6 | 7 | | 3 | 4 | 5 | | 1 | 2 |

8<9 and 8<10, 8 merged, 3<4 and 3<5, 3 is merged. 4 comparison made

Step7: 6 7 8 9 10 1 2 3 4 5

2 comparison, no swap.

Step8: 1 2 3 4 5 6 7 8 9 10

1 comparison made.

In total we sort our array with 11 comparisons and no swap.

B= 1 2 3 4 5 6 7 8 9 10

C = 5 2 13 9 1 7 6 8 1 15 4 11

Step1: 5 2 13 9 1 7 6 8 1 15 4 11

Step2: 5 2 13 9 1 7 6 8 1 15 4 11

Step3: 5 2 13 9 1 7 6 8 1 15 4 11

Step4: 5 2 13 9 1 7 6 8 1 15 4 11

Step5: 2 5 13 1 9 7 6 8 1 4 15 11

2<5 ,1<9, 4<15 ,6<8 ; 4 comparison made.

Step6: 2 5 13 1 7 9 1 6 8 4 11 15

2<13, 5<15, 1<7, 7<9, 1<6, 4<11 and 11<15 ; 7 comparisons and 2 swap

Step7: 1 2 5 7 9 13 1 4 6 8 11 15

Comparisons: 1<2, 2<7, 5<7, 7<13, 9<13 and 1<4, 4<6, 6<11, 8<11, 11<15

10 comparisons,3 swap.

Step8: 1 1 2 4 5 6 7 8 9 11 13 15

10 comparison.

In total we sort array with 31 comparisons and 5 swaps.

C = 1 1 2 4 5 6 7 8 9 11 13 15

D = | S | B | I | M | H | Q | C | L | R | E | P | K |

Step1: | S | B | I | M | H | Q | | C | L | R | E | P | K |

Step2: | S | B | I | | M | H | Q | | C | L | R | | E | P | K |

Step3: | S | B | | I | | M | H | Q | | C | L | | R | | E | P | | K |

Step4: | S | | B | | I | | M | H | Q | | C | L | R | | E | | P | | K |

Step5: | B | S | | I | | H | M | Q | | C | L | | R | | E | P | | K |

B<S, H<M, C<L, E<P; 4 comparison.

Step6: | B | I | S | | H | M | Q | | C | L | R | | E | K | P |

B<I, I<S, Q>M , R>L, E<K, K<P; 6 comparison,2 swap.(for I and K)

Step7: | B | H | I | M | Q | S | | C | E | K | L | P | R |

B<H, H<I, I<M, M<Q, Q<S and C<E, E<L, L>K, L<P, 9 comparison.

Step8: | B | C | E | H | I | K | L | M | P | Q | R | S |

B<C, C<H, H>E, H<K, K>I, K<M, M>L, M<P, P<Q, Q<R, R<S 11 comparison.

D = | **B** | **C** | **E** | **H** | **I** | **K** | **L** | **M** | **P** | **Q** | **R** | **S** |

We sort array with 30 comparisons.

Quick Sort

For quick sort we should choose a pivot value and compare it with two pointers (i used left-l and right-r). After comparison done, our pivot will be in correct position and we will continue the same operations on new subarrays until array is sorted.

Small pseudocode for quick sort that i used for sorting:

İf(arr[l]>arr[pivot]) stop

Else move l++

İf(arr[r]<arr[pivot]) stop

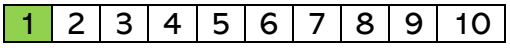
Else move r—

İf(both r and l stopped but l<r) swap arr[l] and arr[r]

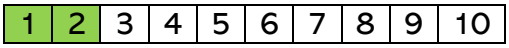
İf(l>=r) swap arr[pivot] and arr[r]

A = | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

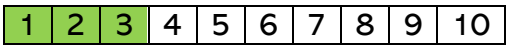
Step1: pivot=1,l=2,r=10, comparison: 9,no swap



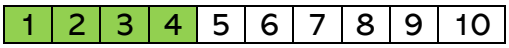
Step2: pivot=2,l=3,r=10, comparison: 8,no swap



Step3: pivot=3,l=4,r=10, comparison: 7,no swap



Step4: pivot=4,l=5,r=10, comparison: 6,no swap



Step5: pivot=5,l=6,r=10, comparison: 5,no swap

Step6: pivot=6,l=7,r=10, comparison: 4,no swap

Step7: pivot=7,l=8,r=10, comparison: 3,no swap

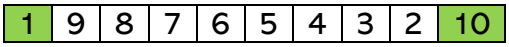
Step8: pivot=8,l=9,r=10, comparison: 2,no swap

Step9: pivot=9,l=10,r=10, comparison: 1,no swap

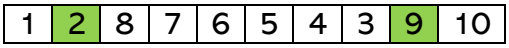
Step10: Array is sorted after 45 comparisons and no swap

B = | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |

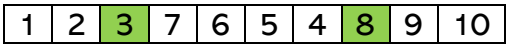
Step1: pivot=10,l=9,r=1, comparison:8, swap10-1



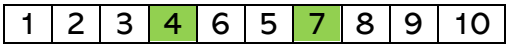
Step2: pivot=9,l=8,r=2, comparison:6, swap 9-2



Step3: pivot=8, l=7,r=3, comparison:4, swap 8-3



Step4: pivot=7,l=6,r=4, comparison:2,swap 7-4

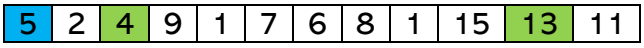


Step5: pivot=6,l=5,r=5,comparison:1, swap 6-5

Array is sorted after 21 comparisons and 5 swap.

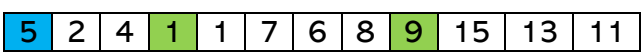
C = | 5 | 2 | 13 | 9 | 1 | 7 | 6 | 8 | 1 | 15 | 4 | 11 |

Step1: pivot=5, l=2,r=11, l stops at 13 and r stops at 4, index of l<r, swap arr[r] -arr[l]. 4 comparison, 1 swap.



Step2: pivot=5,l=4,r=13, l stops at 9,r stops at 1,

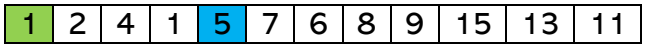
İndex of l<r swap arr[r] -arr[l]. 5 comparison, 1 swap.



Step3: pivot=5,l=1, r=9,l stops at 7,r stops at 1,

İndex of l>=r, swap arr[pivot]-arr[r].Now, pivot at its correct position.

6 comparison,1 swap.



Step4: Because of pivot is in correct position, now we do the same things on subarrays.

Subarr1

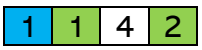
| 1 | 2 | 4 | 1 |

| 5 |

Subarr2

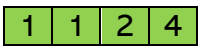
| 7 | 6 | 8 | 9 | 15 | 13 | 11 |

Step5:Subarr1; pivot=1,l=2,r=1 swap Subarr1[l]-Subarr1[r]



2 comparison,1 swap.

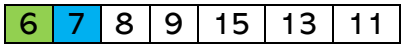
Step6:Subarr1; pivot=1, 1 comparison, 1 swap.



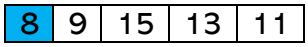
Left side of array C is sorted now.

Step7:Subarr2: pivot=7, l=6,r=11. l stopped at 8,r stopped at 6.

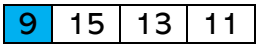
İndex l>=r, swap subarr2[pivot]-subarr2[r].8 comparisons 1 swap.



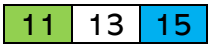
Step8:Subarr3; pivot=8, l=9,r=11, 4 comparisons no swap.



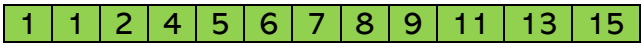
Step9:Subarr4; pivot=9, l=15,r=11,3 comparisons no swap.

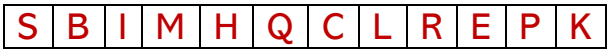


Step10:Subarr5; pivot=15, l=13,r=11,2 comparisons,1 swap.

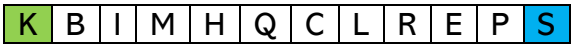


All subarrays sorted using quick sort algorithm in 35 comparisons and 7 swaps.

C= 

D = 

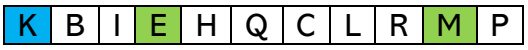
Step1: pivot=S, l=B, r=K, after comparisons l stops at K



Swap arr[pivot]-arr[r]. 1 swap,12 comparisons.

Step2:S is in correct position now, we look left side of S.

Pivot=K, l=B, r=P, l stops at M, r stops at E, l<r swap l-r.



3 comparison 1 swap.

Step3:Pivot=K, l=E ,r=M, l stopped at Q, r stopped at C,

Since l<r swap l-r. 5 comparison 1 swap.

Step4: Pivot=K, l=C, r= Q, l stops at Q, r stops at C,

Swap pivot-r. 2 comparison 1 swap.

Now, pivot is correct position, we will sort left and right sides of pivot using quick sort.

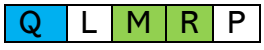
Step5: Pivot=C, l=B, r= H. L stop: l, r stop: B. Swap l and pivot. 6 comparison,1 swap

Step6: Pivot=I, l=E, r=H. L stop: H r stop: E, l>=r swap pivot-r. 2 comparison 1 swap.

Step7: Pivot=I, l and r= H, 1 comparison 1 swap.

Step8: Pivot=Q, l=L, r=P, l stopped: R, r stopped: M, swap l-r.

4 comparison,1 swap.



Step9: Pivot=Q, l=M, r=R, l stopped: R, r stopped: M. swap r and pivot

M	L	Q	R	P

2 comparison,1 swap.

Step10: Q is in correct position, look its right and left,2 comparison 2 swap.

L	M	Q	P	R

We sorted all subarrays and our array is sorted after 37 comparisons and 11 swap.

D=

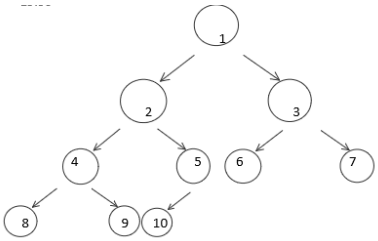
B	C	E	H	I	K	L	M	P	Q	R	S

Heap Sort

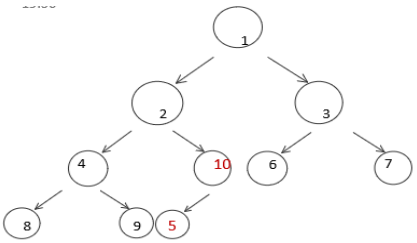
A =

1	2	3	4	5	6	7	8	9	10

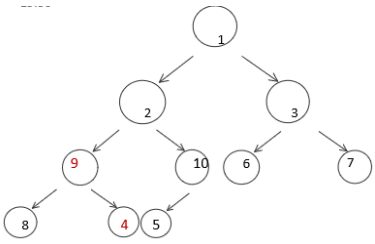
For heap sort, we need to change our arrays as max heap and then we will delete roots as maximum element and we will add them to end of the array, this way we can sort ı-our arrays as increasing order.



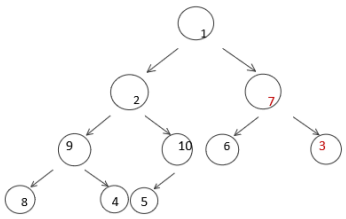
First version of tree. We need to turn it to a max heap.



10-5 swapped.

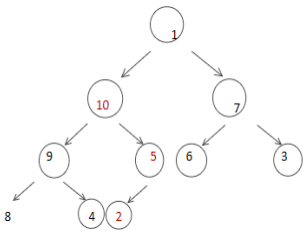


9-4 swapped.



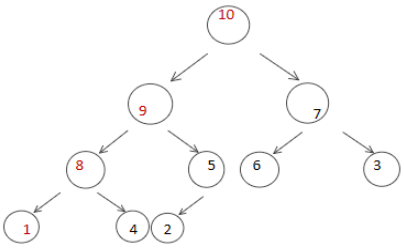
8!>9 no swap, 7>3 swap 7-3.

6!>7 no swap.



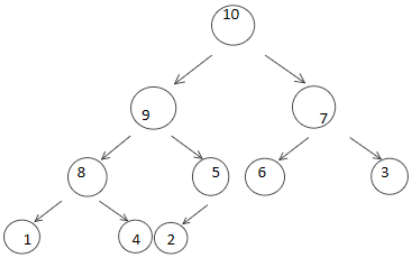
10>2, swap 10-2,and

5>2, swap 5-2



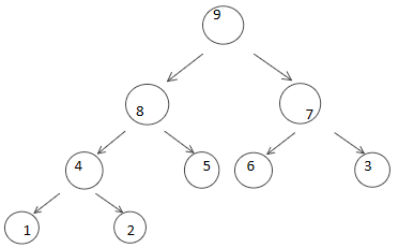
10>1, 9>1,8>1 swap 10-1, 9-1 ,8-1. We reached max heap.

After reaching max heap, we will delete max element from heap, rearrenge the heap and delete max element again until array is sorted.



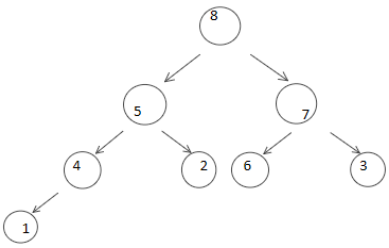
We will delete 10 and place it to last element in our array. Then we will make 2(latest child) as root and we will compare it with 9,8,4, since all of them are bigger than 2 we will make 3 comparisons and 3 swaps.
Current array:

9	8	7	4	5	6	3	1	2	10



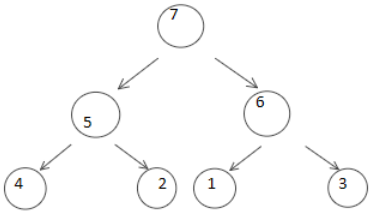
Delete 9 and make 2 root, 2<8 swap 2-8, 2<5 swap 2-5 .
2 comparisons 2 swap.
Current array:

8	5	7	4	2	6	3	1	9	10



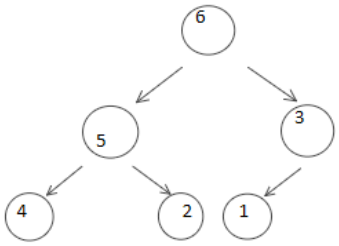
We will delete 8 and make 1 root, then 1<7 swap 1-7, 1<6 swap 1-6, 4 comparisons 4 swap.
Current array:

7	5	6	4	2	1	3	8	9	10



Remove 7 , 3 is new root, compare 3 with 6,swap them.
3 comparison 1 swap
Current array:

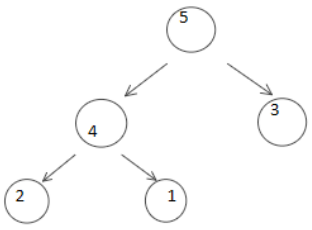
6	5	3	4	2	1	7	8	9	10



Remove 6, make 1 root, swap 1-5 and 1-4. 2 comparison 2 swap.

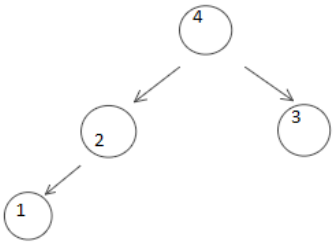
Current array:

5	4	3	1	2	6	7	8	9	10



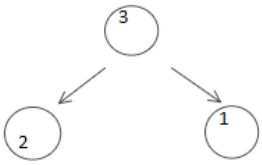
Remove 5, make 1 root, swap 1-4, swap 1-2. 3 comparison 2 swap.
Current array:

4	2	3	1	5	6	7	8	9	10



Remove 4, 1 is new root, swap 1-3. 2 comparison 1 swap.
Current array:

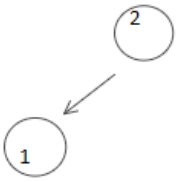
3	2	1	4	5	6	7	8	9	10



Remove 3, make 1 root, swap 1-2 . 1 comparison 1 swap.

Current array:

2	1	3	4	5	6	7	8	9	10



Remove 2, swap 1 as root, 1 swap.

Current array:

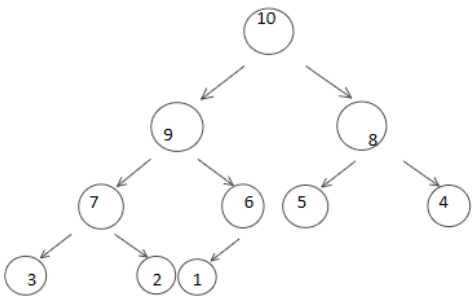
1	2	3	4	5	6	7	8	9	10

Remove 1, no comparison no swap . Array is sorted:

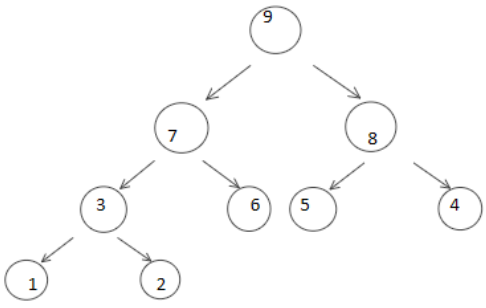
1	2	3	4	5	6	7	8	9	10

B =

10	9	8	7	6	5	4	3	2	1

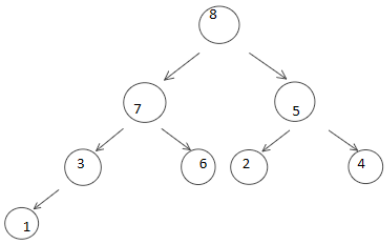


When we create a heap, we can see that array is alredy in max heap shape. Now we will remove root and rearrenge the heap until array is sorted.



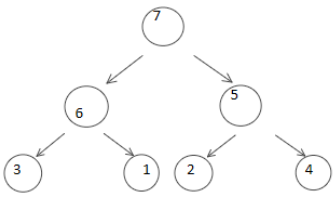
10 is removed, 1 new root, swap 1 with 9,7,3.
6 comparison, 3 swap.
Current array:

9	7	8	3	6	5	4	1	2	10



9 removed, 2 is root, swap 2 with 8,5 , 4 comparison 2 swap.
Current array:

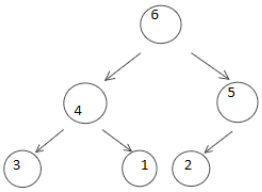
8	7	5	3	6	2	4	1	9	10



8 removed, 1 is root, swap 1 with 7 and 6, 4 comparison 2 swap.

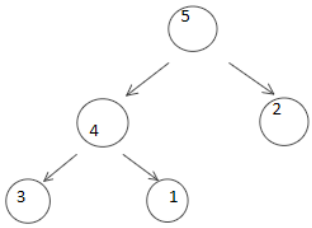
Current array:

7	6	5	3	1	2	4	8	9	10



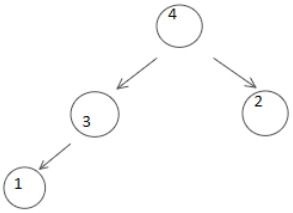
7 removed, 4 is root, since 6 >4 swap 4-6. 2 comparison,1 swap.
Current array:

6	4	5	3	1	2	7	8	9	10



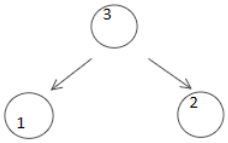
6 removed,2 is new root, swap 2- 5. 2comparison 1 swap.
Current array:

5	4	2	3	1	6	7	8	9	10



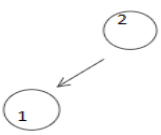
5 removed,1 is new root, swap 1-4. 2 comparison 1 swap.
Current array:

4	3	2	1	5	6	7	8	9	10



4 removed,1 is new root, swap 1-3.
Current array:

3	1	2	4	5	6	7	8	9	10



3 removed,2 is new root, no swap.

2 removed, 1 is new root

Current array:

2	1	3	4	5	6	7	8	9	10

1	2	3	4	5	6	7	8	9	10

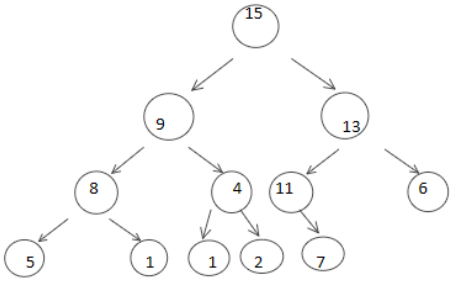
Array is sorted.

C =

5	2	13	9	1	7	6	8	1	15	4	11

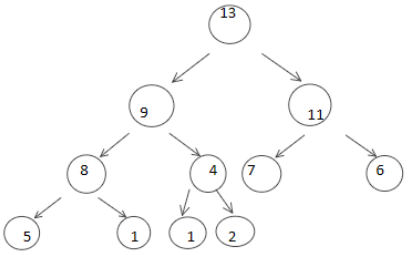
We will rearrenge our array first, to reach a max heap.

Swap 11- 7,swap 15-1, and 15-2, swap 15-5, then swap 5-9 and 5-8. We reached max heap after 6 swap. Now we can remove max numbers and sort array.



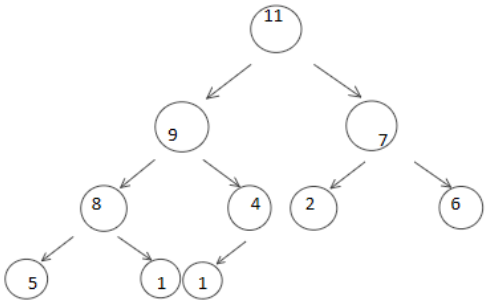
Max heap
Current array:

15	9	13	8	4	11	6	5	1	1	2	7



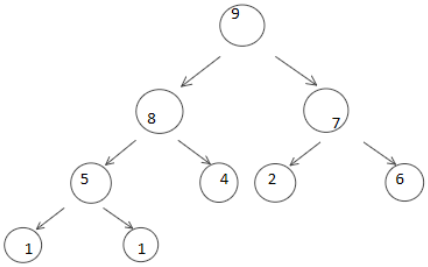
15 removed, 7 new root, , 11-7 and 11-13 swapped. 4 comparison, 2swap.
Current array:

13	9	11	8	4	7	6	5	1	1	2	15



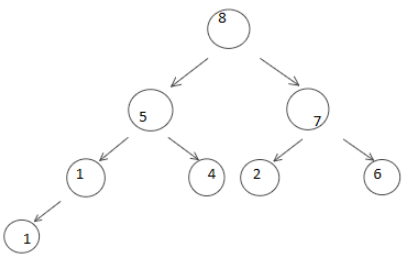
13 removed, 2 new root, 2-11 swap, 2-7 swap, 4 comparison 2 swap.
Current array:

11	9	7	8	4	2	6	5	1	1	13	15



11 removed, 1 new root, 1-9, 1-8, 1-5 swapped. 5 comparison,3 swap.
Current array:

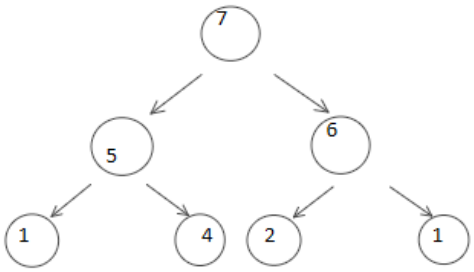
9	8	7	5	4	2	6	1	1	11	13	15



9 removed,1 new root, 1-8 and 1-5 swapped. 4 comparison 2 swap.

Current array:

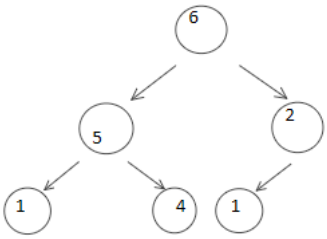
8	5	7	1	4	2	6	1	9	11	13	15



8 removed, 1new root, 1-7,1-6 swap. 4 comparison 2 swap.

Current array:

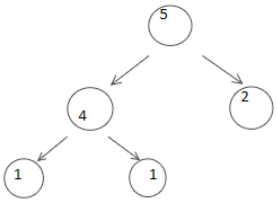
7	5	6	1	4	2	1	8	9	11	13	15



7 removed, 1 new root, 1-6 swap,1-2 swap, 4 comparison 2 swap.

Current array:

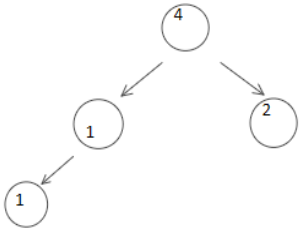
6	5	2	1	4	1	7	8	9	11	13	15



6 removed, 1 new root, 1-5 swap, 1-4 swap. 4 comparison 2 swap.

Current array:

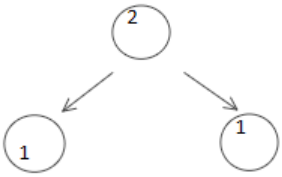
5	4	2	1	1	6	7	8	9	11	13	15



5 removed, 1 new root, 1-4 swap. 2 comparison 1 swap.

Current array:

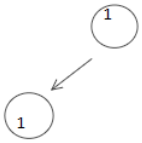
4	1	2	1	5	6	7	8	9	11	13	15



4 removed, 1 new root, 1-2 swap. 2 comparison 1 swap.

Current array :

2	1	1	4	5	6	7	8	9	11	13	15



2 removed, 1 is new root. 1 comparison.

Current array:

1	1	2	4	5	6	7	8	9	11	13	15

1 removed, 1 is new root. Current array:

1	1	2	4	5	6	7	8	9	11	13	15

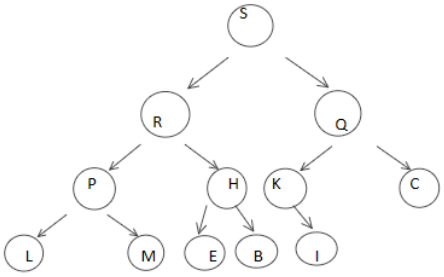
Array is sorted.

1	1	2	4	5	6	7	8	9	11	13	15

D =

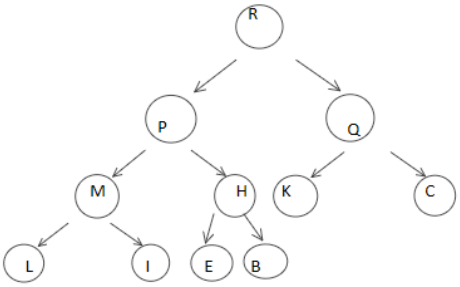
S	B	I	M	H	Q	C	L	R	E	P	K

To reach max heap, we need to make some changes. Swap I-Q and I-K, P-H and P-B , swap R-M, R-P. 6 swap is made for reaching max heap. Now we will sort our array using max heap.



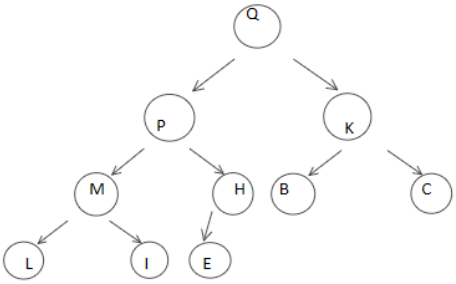
Max heap.
Current array:

S	R	Q	P	H	K	C	L	M	E	B	I



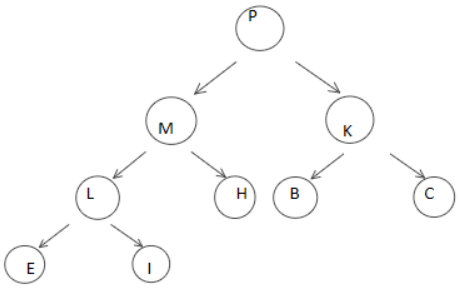
S removed, I new root, I-R, I-P, I-M swapped.
5 comparison, 3 swap.
Current array:

R	P	Q	M	H	K	C	L	I	E	B	S



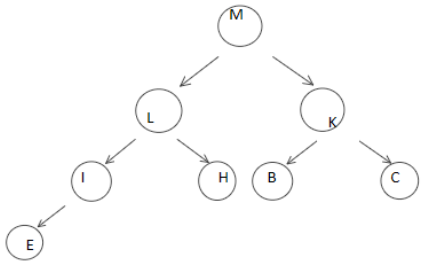
R removed. B new root, B-Q, B-K swap.
4 comparison,2 swap.
Current array:

Q	P	K	M	H	B	C	L	I	E	R	S



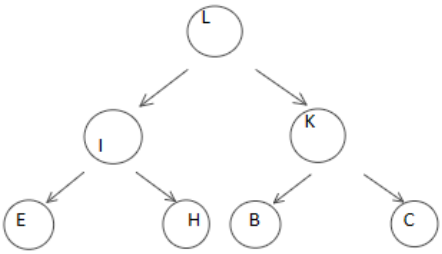
Q removed, E new root, E-P, E-M,E-L swap.
6 comparison 3 swap.
Current array:

P	M	K	L	H	B	C	E	I	Q	R	S



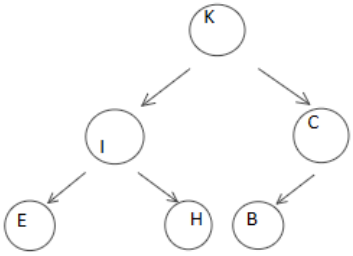
P removed, I new root, I-M and I-L swap.
4 comparison,2 swap.
Current array:

M	L	K	I	H	B	C	E	P	Q	R	S



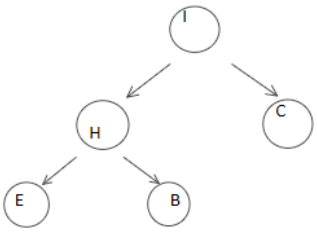
M removed, E new root, E-L, E-I swap.
4 comparison,2 swap.
Current array:

L	I	K	E	H	B	C	M	P	Q	R	S



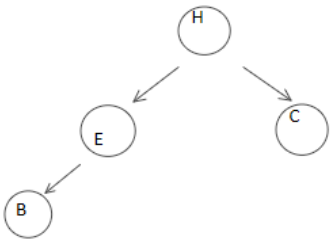
L removed, C is new root, swap C-K. 2 comparison, 1 swap.
Current array:

K	I	C	E	H	B	L	M	P	Q	R	S



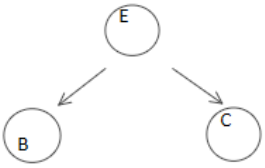
K removed, B new root. Swap B-I and B-E.
4 comparison,2 swap.
Current array:

I	H	C	E	B	K	L	M	P	Q	R	S



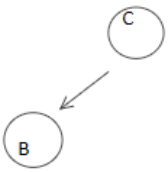
I removed, B new root, swap B-H, B-E. 4 comparison 2 swap.
Current array:

H	E	C	B	I	K	L	M	P	Q	R	S



H removed. B is new root, B-e swap. 2 comparison,1 swap.
Current array:

E	B	C	H	I	K	L	M	P	Q	R	S



E removed, C is new new root.
Current array:

C	B	E	H	I	K	L	M	P	Q	R	S

C removed. B is the root.
Current array:

B	C	E	H	I	K	L	M	P	Q	R	S

Array is sorted.

B	C	E	H	I	K	L	M	P	Q	R	S