GIT Department of Computer Engineering

CSE 222/505 – Spring 2020

Homework #06 Part 1 Report

Abdullah ÇELİK

171044002

Q1:

- A is an ordered integer array with 10 elements from small to large
  A = {0,1,2,3,4,5,6,7,8,9}

  Shell Sort:
  pass = 1
  gap = floor(size/2) = floor(10/2) = floor(5) = 5

  It is made a **virtual** sub-list of all values located at the interval of gap positions.
  {0,5},{1,6},{2,7},{3,8},{4,9}

  It is compared values in each sub-list and swap them (if necessary) in the original array.
  0 > 5 ? false
  1 > 6 ? false
  2 > 7 ? false
  3 > 8 ? false
  4 > 9 ? false

  Then calculate gap again.
  pass = 2
  gap = floor(gap/2) = floor(5/2) = floor(2,5) = 2

  A = {0,1,2,3,4,5,6,7,8,9}
  It is made a **virtual** sub-list of all values located at the interval of gap positions.
  {0,2,4,6,8},{1,3,5,7,9}

  It is compared values in each sub-list and swap them (if necessary) in the original array.
  0 > 2 ? false
  1 > 3 ? false
  2 > 4 ? false
  3 > 5 ? false
  4 > 6 ? false
  5 > 7 ? false
  6 > 8 ? false
  7 > 9 ? false

  Then calculate gap again.
  pass = 3
  gap = floor(gap/2) = floor(2/2) = floor(1) = 1

  A = {0,1,2,3,4,5,6,7,8,9}
  Finally, we sort the rest of the array using interval of value 1. Shell sort uses insertion sort to sort the array.
  0 > 1 ? false
  1 > 2 ? false
  2 > 3 ? false
  3 > 4 ? false

4 > 5 ? false
5 > 6 ? false
6 > 7 ? false
7 > 8 ? false
8 > 9 ? false

Then calculate gap again
gap = floor(gap/2) = floor(1/2) = floor(0,5) = 0
And shell sort algroitm is over. Array was sorted.

Total comparision : 22
Total displacement : 0

Merge Sort:

<table>
<tr><td colspan="10">0,1,2,3,4,5,6,7,8,9</td></tr>
<tr><td colspan="10">Size = 10
l(left) = 0
r(right) = size -1 = 9
l < r, divide array by two
m(middle) = floor((l+r)/2) = 4</td></tr>
<tr><td colspan="5">0,1,2,3,4</td><td colspan="5">5,6,7,8,9</td></tr>
<tr><td colspan="5">l = 0
r = 4
l < r
m = 2</td><td colspan="5">l = 5
r = 9
l < r
m = 7</td></tr>
<tr><td colspan="3">0,1,2</td><td colspan="2">3,4</td><td colspan="3">5,6,7</td><td colspan="2">8,9</td></tr>
<tr><td colspan="3">l = 0
r = 2
l < r
m = 1</td><td colspan="2">l = 3
r = 4
l < r
m = 3</td><td colspan="3">l = 5
r = 7
l < r
m = 6</td><td colspan="2">l = 8
r = 9
l < r
m = 8</td></tr>
<tr><td colspan="2">0,1</td><td>2</td><td>3</td><td>4</td><td colspan="2">5,6</td><td>7</td><td>8</td><td>9</td></tr>
<tr><td colspan="2">l = 0
r = 1
l < r
m = 0</td><td>l = 2
r = 2
l >= r</td><td>l = 2
r = 2
l >= r</td><td>l = 4
r = 4
l >= r</td><td colspan="2">l = 5
r = 6
l < r
m = 5</td><td>l = 7
r = 7
l >= r</td><td>l = 8
r = 8
l >= r</td><td>l = 9
r = 9
l >= r</td></tr>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td></tr>
<tr><td>l = 0
r = 0
l >= r</td><td>l = 1
r = 1
l >= r</td><td></td><td></td><td></td><td>l = 5
r = 5
l >= r</td><td>l = 6
r = 6
l >= r</td><td></td><td></td><td></td></tr>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td></tr>
<tr><td colspan="2">0 < 1 ? true</td><td></td><td></td><td></td><td colspan="2">5 < 6 ? true</td><td></td><td></td><td></td></tr>
<tr><td colspan="2">0,1</td><td>2</td><td>3</td><td>4</td><td colspan="2">5,6</td><td>7</td><td>8</td><td>9</td></tr>
<tr><td colspan="3">0 < 2 ? true
1 < 2 ? true</td><td colspan="2">3 < 4 ? true</td><td colspan="3">5 < 7 ? true
6 < 7 ? true</td><td colspan="2">8 < 9 ? true</td></tr>
<tr><td colspan="3">0,1,2</td><td colspan="2">3,4</td><td colspan="3">5,6,7</td><td colspan="2">8,9</td></tr>
<tr><td colspan="5">0 < 3 ? true
1 < 3 ? true
2 < 3 ? true</td><td colspan="5">5 < 8 ? true
6 < 8 ? true
7 < 8 ? true</td></tr>
<tr><td colspan="5">0,1,2,3,4</td><td colspan="5">5,6,7,8,9</td></tr>
</table>

0 < 5 ? true 1 < 5 ? true 2 < 5 ? true 3 < 5 ? true 4 < 5 ? true
0,1,2,3,4,5,6,7,8,9

Total comparision : 38 (base case + comprision between array elements)
Total displacement : 34 (replace elements a new array)

Heap Sort:

A = {0,1,2,3,4,5,6,7,8,9}				
Build a Max Heap array				
Algorithm: In the default array, the value corresponding to the array index is compared to the value corresponding to the parent index ((array index-1) / 2) in the max heap array. If the operation is true, their positions are swapped. This process continues until the comparision operation is false or has no parent. The general process continues until array index is smaller than size.				
Array Index	**Max Heap Array**	**Parent Index**	**Description**	**After operation Max Heap Array**
0	0,1,2,3,4,5,6,7,8,9	-	Initially 0. index is root	0,1,2,3,4,5,6,7,8,9
1	0,1,2,3,4,5,6,7,8,9	0	1 > 0 ? true – swap them	1,0,2,3,4,5,6,7,8,9
1	1,0,2,3,4,5,6,7,8,9	-	Element 1 doesn't have parent	1,0,2,3,4,5,6,7,8,9
2	1,0,2,3,4,5,6,7,8,9	0	2 > 1 ? true – swap them	2,0,1,3,4,5,6,7,8,9
2	1,0,2,3,4,5,6,7,8,9	-	Element 2 doesn't have parent	2,0,1,3,4,5,6,7,8,9
3	2,0,1,3,4,5,6,7,8,9	1	3 > 0 ? true – swap them	2,3,1,0,4,5,6,7,8,9
3	2,3,1,0,4,5,6,7,8,9	0	3 > 2 ? true – swap them	3,2,1,0,4,5,6,7,8,9
3	3,2,1,0,4,5,6,7,8,9	-	Element 3 doesn't have parent	3,2,1,0,4,5,6,7,8,9
4	3,2,1,0,4,5,6,7,8,9	1	4 > 2 ? true – swap them	3,4,1,0,2,5,6,7,8,9
4	3,4,1,0,2,5,6,7,8,9	0	4 > 3 ? true – swap them	4,3,1,0,2,5,6,7,8,9
4	4,3,1,0,2,5,6,7,8,9	-	Element 4 doesn't have parent	4,3,1,0,2,5,6,7,8,9
5	4,3,1,0,2,5,6,7,8,9	2	5 > 1 ? true – swap them	4,3,5,0,2,1,6,7,8,9
5	4,3,5,0,2,1,6,7,8,9	0	5 > 4 ? true – swap them	5,3,4,0,2,1,6,7,8,9
5	5,3,4,0,2,1,6,7,8,9	-	Element 5 doesn't have parent	5,3,4,0,2,1,6,7,8,9
6	5,3,4,0,2,1,6,7,8,9	2	6 > 4 ? true – swap them	5,3,6,0,2,1,4,7,8,9
6	5,3,6,0,2,1,4,7,8,9	0	6 > 5 ? true – swap them	6,3,5,0,2,1,4,7,8,9
6	6,3,5,0,2,1,4,7,8,9	-	Element 6 doesn't have parent	6,3,5,0,2,1,4,7,8,9
7	6,3,5,0,2,1,4,7,8,9	3	7 > 0 ? true – swap them	6,3,5,7,2,1,4,0,8,9
7	6,3,5,7,2,1,4,0,8,9	1	7 > 3 ? true – swap them	6,7,5,3,2,1,4,0,8,9
7	6,7,5,3,2,1,4,0,8,9	0	7 > 6 ? true – swap them	7,6,5,3,2,1,4,0,8,9
7	7,6,5,3,2,1,4,0,8,9	-	Element 7 doesn't have parent	7,6,5,3,2,1,4,0,8,9
8	7,6,5,3,2,1,4,0,8,9	3	8 > 3 ? true – swap them	7,6,5,8,2,1,4,0,3,9
8	7,6,5,8,2,1,4,0,3,9	1	8 > 6 ? true – swap them	7,8,5,6,2,1,4,0,3,9
8	7,8,5,6,2,1,4,0,3,9	0	8 > 7 ? true – swap them	8,7,5,6,2,1,4,0,3,9
8	8,7,5,6,2,1,4,0,3,9	-	Element 8 doesn't have parent	8,7,5,6,2,1,4,0,3,9
9	8,7,5,6,2,1,4,0,3,9	4	9 > 2 ? true – swap them	8,7,5,6,9,1,4,0,3,2
9	8,7,5,6,9,1,4,0,3,2	1	9 > 7 ? true – swap them	8,9,5,6,7,1,4,0,3,2
9	8,9,5,6,7,1,4,0,3,2	0	9 > 8 ? true – swap them	9,8,5,6,7,1,4,0,3,2

9	9,8,5,6,7,1,4,0,3,2	-	Element 9 doesn't have parent	9,8,5,6,7,1,4,0,3,2
10			Max heap array is created	9,8,5,6,7,1,4,0,3,2

A = {9,8,5,6,7,1,4,0,3,2}			
Sorting			
Algorithm: Swap array index with head in array. Heapify is applied to the part up to the array index in the array. Sorting is done until the array index is 0.			
Array Index	**Array**	**Description**	**After operation array**
9	9,8,5,6,7,1,4,0,3,2	Head and array index were swapped. Apply heapify	2,8,5,6,7,1,4,0,3,9
	2,8,5,6,7,1,4,0,3,9	8 > 5 ? true max child = 8 2 < 8 ? true – swap them	8,2,5,6,7,1,4,0,3,9
	8,2,5,6,7,1,4,0,3,9	6 > 7 ? false max child = 7 2 < 7 ? true – swap them	8,7,5,6,2,1,4,0,3,9
	8,7,5,6,2,1,4,0,3,9	Element 2 doesn't have children	8,7,5,6,2,1,4,0,3,9
8	8,7,5,6,2,1,4,0,3,9	Head and array index were swapped. Apply heapify	3,7,5,6,2,1,4,0,8,9
	3,7,5,6,2,1,4,0,8,9	7 > 5 ? true max child = 7 3 < 7 ? true – swap them	7,3,5,6,2,1,4,0,8,9
	7,3,5,6,2,1,4,0,8,9	6 > 2 ? true max child = 6 3 < 6 ? true – swap them	7,6,5,3,2,1,4,0,8,9
	7,6,5,3,2,1,4,0,8,9	Max child = 0 3 < 0 ? false	7,6,5,3,2,1,4,0,8,9
7	7,6,5,3,2,1,4,0,8,9	Head and array index were swapped. Apply heapify	0,6,5,3,2,1,4,7,8,9
	0,6,5,3,2,1,4,7,8,9	6 > 5 ? true max child = 6 0 < 6 ? true – swap them	6,0,5,3,2,1,4,7,8,9
	6,0,5,3,2,1,4,7,8,9	3 > 2 ? true max child = 3 0 < 3 ? true – swap them	6,3,5,0,2,1,4,7,8,9
	6,3,5,0,2,1,4,7,8,9	Element 0 doesn't have children	6,3,5,0,2,1,4,7,8,9
6	6,3,5,0,2,1,4,7,8,9	Head and array index were swapped. Apply heapify	4,3,5,0,2,1,6,7,8,9
	4,3,5,0,2,1,6,7,8,9	3 > 5 ? false max child = 5 4 < 5 ? true – swap them	5,3,4,0,2,1,6,7,8,9
	5,3,4,0,2,1,6,7,8,9	Max child = 1 4 < 1 false	5,3,4,0,2,1,6,7,8,9
5	5,3,4,0,2,1,6,7,8,9	Head and array index were swapped. Apply heapify	1,3,4,0,2,5,6,7,8,9
	1,3,4,0,2,5,6,7,8,9	3 > 4 ? false max child = 4 1 < 4 ? true – swap them	4,3,1,0,2,5,6,7,8,9
	4,3,1,0,2,5,6,7,8,9	0 > 2 ? false max child = 2 1 < 2 ? true – swap them	4,3,2,0,1,5,6,7,8,9
	4,3,2,0,1,5,6,7,8,9	Element 1 doesn't have children	4,3,2,0,1,5,6,7,8,9
4	4,3,2,0,1,5,6,7,8,9	Head and array index were swapped. Apply heapify	1,3,2,0,4,5,6,7,8,9
	1,3,2,0,4,5,6,7,8,9	3 > 2 ? true max child = 3 1 < 3 ? true – swap them	3,1,2,0,4,5,6,7,8,9
	3,1,2,0,4,5,6,7,8,9	Max child = 0	3,1,2,0,4,5,6,7,8,9

		1 < 0 ? false	
3	3,1,2,0,4,5,6,7,8,9	Head and array index were swapped. Apply heapify	0,1,2,3,4,5,6,7,8,9
	0,1,2,3,4,5,6,7,8,9	1 > 2 ? false max child = 2 0 < 2 ? true – swap them	2,1,0,3,4,5,6,7,8,9
	2,1,0,3,4,5,6,7,8,9	Element 0 doesn't have children	2,1,0,3,4,5,6,7,8,9
2	2,1,0,3,4,5,6,7,8,9	Head and array index were swapped. Apply heapify	0,1,2,3,4,5,6,7,8,9
	0,1,2,3,4,5,6,7,8,9	Max child = 1 0 < 1 ? true – swap them	1,0,2,3,4,5,6,7,8,9
	1,0,2,3,4,5,6,7,8,9	Element 0 doesn't have children	1,0,2,3,4,5,6,7,8,9
1	1,0,2,3,4,5,6,7,8,9	Head and array index were swapped.	0,1,2,3,4,5,6,7,8,9
	0,1,2,3,4,5,6,7,8,9	Element 0 doesn't have children	0,1,2,3,4,5,6,7,8,9
0	0,1,2,3,4,5,6,7,8,9	Array is sorted	0,1,2,3,4,5,6,7,8,9

Total comparision : 64 (build + sort)
Total displacement : 40 (swap amount)

Quick Sort:

Pivot: Always picked last element as pivot
Partition Algorithm
The logic is simple, we start from the leftmost element and keep track of index of smaller elements as i. While traversing, if we find a smaller element, we swap current element with arr[i]. Otherwise we ignore current element.

A = {0,1,2,3,4,5,6,7,8,9}

Array	Pivot Value	p	i	end	Description	After operation array
0,1,2,3,4,5,6,7,8,9			0	9	i < end ? true – partition calls.	
0,1,2,3,4,5,6,7,8,9	9	-1	0	9	0 < 9 ? true, ++p, swap p and i (0,0)	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9	9	0	1	9	1 < 9 ? true, ++p, swap p and i (1,1)	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9	9	1	2	9	2 < 9 ? true, ++p, swap p and i (2,2)	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9	9	2	3	9	3 < 9 ? true, ++p, swap p and i (3,3)	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9	9	3	4	9	4 < 9 ? true, ++p, swap p and i (4,4)	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9	9	4	5	9	5 < 9 ? true, ++p, swap p and i (5,5)	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9	9	5	6	9	6 < 9 ? true, ++p, swap p and i (6,6)	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9	9	6	7	9	7 < 9 ? true, ++p, swap p and i (7,7)	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9	9	7	8	9	8 < 9 ? true, ++p, swap p and i (8,8)	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9	9	8	9	9	i < end ? false. Last operation ++p and swap p and pivot (9,9). Partition is finish. So next recursion calls Left part: i = 0, end(p-1) = 8 Right part: i(p+1) = 10, end = 9	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9			0	8	i < end ? true – partition calls.	
0,1,2,3,4,5,6,7,8,9	8	-1	0	8	0 < 8 ? true, ++p, swap p and i (0,0)	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9	8	0	1	8	1 < 8 ? true, ++p, swap p and i (1,1)	0,1,2,3,4,5,6,7,8,9

0,1,2,3,4,5,6,7,8,9	8	1	2	8	2 < 8 ? true, ++p, swap p and i (2,2)	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9	8	2	3	8	3 < 8 ? true, ++p, swap p and i (3,3)	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9	8	3	4	8	4 < 8 ? true, ++p, swap p and i (4,4)	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9	8	4	5	8	5 < 8 ? true, ++p, swap p and i (5,5)	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9	8	5	6	8	6 < 8 ? true, ++p, swap p and i (6,6)	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9	8	6	7	8	7 < 8 ? true, ++p, swap p and i (7,7)	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9	8	7	8	8	i < end ? false. Last operation ++p and swap p and pivot (8,8). Partition is finish. So next recursion calls Left part: i = 0, end(p-1) = 7 Right part: i(p+1) = 9, end = 8	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9			0	7	i < end ? true – partition calls.	
0,1,2,3,4,5,6,7,8,9	7	-1	0	7	0 < 7 ? true, ++p, swap p and i (0,0)	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9	7	0	1	7	1 < 7 ? true, ++p, swap p and i (1,1)	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9	7	1	2	7	2 < 7 ? true, ++p, swap p and i (2,2)	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9	7	2	3	7	3 < 7 ? true, ++p, swap p and i (3,3)	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9	7	3	4	7	4 < 7 ? true, ++p, swap p and i (4,4)	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9	7	4	5	7	5 < 7 ? true, ++p, swap p and i (5,5)	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9	7	5	6	7	6 < 7 ? true, ++p, swap p and i (6,6)	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9	7	6	7	7	i < end ? false. Last operation ++p and swap p and pivot (7,7). Partition is finish. So next recursion calls Left part: i = 0, end(p-1) = 6 Right part: i(p+1) = 8, end = 7	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9			0	6	i < end ? true – partition calls.	
0,1,2,3,4,5,6,7,8,9	6	-1	0	6	0 < 6 ? true, ++p, swap p and i (0,0)	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9	6	0	1	6	1 < 6 ? true, ++p, swap p and i (1,1)	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9	6	1	2	6	2 < 6 ? true, ++p, swap p and i (2,2)	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9	6	2	3	6	3 < 6 ? true, ++p, swap p and i (3,3)	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9	6	3	4	6	4 < 6 ? true, ++p, swap p and i (4,4)	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9	6	4	5	6	5 < 6 ? true, ++p, swap p and i (5,5)	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9	6	5	6	6	i < end ? false. Last operation ++p and swap p and pivot (6,6). Partition is finish. So next recursion calls Left part: i = 0, end(p-1) = 5 Right part: i(p+1) = 7, end = 6	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9			0	5	i < end ? true – partition calls.	
0,1,2,3,4,5,6,7,8,9	5	-1	0	5	0 < 5 ? true, ++p, swap p and i (0,0)	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9	5	0	1	5	1 < 5 ? true, ++p, swap p and i (1,1)	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9	5	1	2	5	2 < 5 ? true, ++p, swap p and i (2,2)	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9	5	2	3	5	3 < 5 ? true, ++p, swap p and i (3,3)	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9	5	3	4	5	4 < 5 ? true, ++p, swap p and i (4,4)	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9	5	4	5	5	i < end ? false. Last operation ++p and swap p and pivot (5,5). Partition is finish. So next recursion calls Left part: i = 0, end(p-1) = 4 Right part: i(p+1) = 6, end = 5	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9			0	4	i < end ? true – partition calls.	
0,1,2,3,4,5,6,7,8,9	4	-1	0	4	0 < 4 ? true, ++p, swap p and i (0,0)	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9	4	0	1	4	1 < 4 ? true, ++p, swap p and i (1,1)	0,1,2,3,4,5,6,7,8,9

0,1,2,3,4,5,6,7,8,9	4	1	2	4	2 < 4 ? true, ++p, swap p and i (2,2)	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9	4	2	3	4	3 < 4 ? true, ++p, swap p and i (3,3)	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9	4	3	4	4	i < end ? false. Last operation ++p and swap p and pivot (4,4). Partition is finish. So next recursion calls Left part: i = 0, end(p-1) = 3 Right part: i(p+1) = 5, end = 4	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9			0	3	i < end ? true – partition calls.	
0,1,2,3,4,5,6,7,8,9	3	-1	0	3	0 < 3 ? true, ++p, swap p and i (0,0)	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9	3	0	1	3	1 < 3 ? true, ++p, swap p and i (1,1)	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9	3	1	2	3	2 < 3 ? true, ++p, swap p and i (2,2)	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9	3	2	3	3	i < end ? false. Last operation ++p and swap p and pivot (3,3). Partition is finish. So next recursion calls Left part: i = 0, end(p-1) = 2 Right part: i(p+1) = 4, end = 3	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9			0	2	i < end ? true – partition calls.	
0,1,2,3,4,5,6,7,8,9	2	-1	0	2	0 < 2 ? true, ++p, swap p and i (0,0)	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9	2	0	1	2	1 < 2 ? true, ++p, swap p and i (1,1)	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9	2	1	2	2	i < end ? false. Last operation ++p and swap p and pivot (2,2). Partition is finish. So next recursion calls Left part: i = 0, end(p-1) = 1 Right part: i(p+1) = 3, end = 2	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9			0	1	i < end ? true – partition calls.	
0,1,2,3,4,5,6,7,8,9	1	-1	0	1	0 < 1 ? true, ++p, swap p and i (0,0)	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9	1	0	1	1	i < end ? false. Last operation ++p and swap p and pivot (1,1). Partition is finish. So next recursion calls Left part: i = 0, end(p-1) = 0 Right part: i(p+1) = 2, end = 1	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9			0	0	i < end ? false	
0,1,2,3,4,5,6,7,8,9			2	1	i < end ? false	
0,1,2,3,4,5,6,7,8,9			3	2	i < end ? false	
0,1,2,3,4,5,6,7,8,9			4	3	i < end ? false	
0,1,2,3,4,5,6,7,8,9			5	4	i < end ? false	
0,1,2,3,4,5,6,7,8,9			6	5	i < end ? false	
0,1,2,3,4,5,6,7,8,9			7	6	i < end ? false	
0,1,2,3,4,5,6,7,8,9			8	7	i < end ? false	
0,1,2,3,4,5,6,7,8,9			9	8	i < end ? false	
0,1,2,3,4,5,6,7,8,9			10	9	i < end ? false	
					Array is sorted	0,1,2,3,4,5,6,7,8,9

Total comparision : 73 (base case + comparision between array elements)
Total displacement : 54 (swap amount)

- B is an ordered integer array with 10 elements from large to small
  B = {9,8,7,6,5,4,3,2,1,0}

  Shell Sort:
  pass = 1
  gap = floor(size/2) = floor(10/2) = floor(5) = 5

  It is made a **virtual** sub-list of all values located at the interval of gap positions.
  {9,4},{8,3},{7,2},{6,1},{5,0}

  It is compared values in each sub-list and swap them (if necessary) in the original array.
  9 > 4 ? true – swap them B = {4,8,7,6,5,9,3,2,1,0}
  8 > 3 ? true – swap them B = {4,3,7,6,5,9,8,2,1,0}
  7 > 2 ? true – swap them B = {4,3,2,6,5,9,8,7,1,0}
  6 > 1 ? true – swap them B = {4,3,2,1,5,9,8,7,6,0}
  5 > 0 ? true – swap them B = {4,3,2,1,0,9,8,7,6,5}

  Then calculate gap again.
  pass = 2
  gap = floor(gap/2) = floor(5/2) = floor(2,5) = 2

  B = {4,3,2,1,0,9,8,7,6,5}
  It is made a **virtual** sub-list of all values located at the interval of gap positions.
  {4,2,0,8,6}{3,1,9,7,5}

  It is compared values in each sub-list and swap them (if necessary) in the original array.
  4 > 2 ? true – swap them B = {2,3,4,1,0,9,8,7,6,5}
  3 > 1 ? true – swap them B = {2,1,4,3,0,9,8,7,6,5}
  4 > 0 ? true – swap them B = {2,1,0,3,4,9,8,7,6,5}
  2 > 0 ? true – swap them B = {0,1,2,3,4,9,8,7,6,5}
  3 > 9 ? false
  4 > 8 ? false
  9 > 7 ? true – swap them B = {0,1,2,3,4,7,8,9,6,5}
  3 > 7 ? false
  8 > 6 ? true – swap them B = {0,1,2,3,4,7,6,9,8,5}
  4 > 6 ? false
  9 > 5 ? true – swap them B = {0,1,2,3,4,7,6,5,8,9}
  7 > 5 ? true – swap them B = {0,1,2,3,4,5,6,7,8,9}

  Then calculate gap again.
  pass = 3
  gap = floor(gap/2) = floor(2/2) = floor(1) = 1

  A = {0,1,2,3,4,5,6,7,8,9}
  Finally, we sort the rest of the array using interval of value 1. Shell sort uses insertion sort to sort the array.
  0 > 1 ? false
  1 > 2 ? false

2 > 3 ? false
3 > 4 ? false
4 > 5 ? false
5 > 6 ? false
6 > 7 ? false
7 > 8 ? false
8 > 9 ? false

Then calculate gap again
gap = floor(gap/2) = floor(1/2) = floor(0,5) = 0
And shell sort algroitm is over. Array was sorted.

Size : 10
Total comparision : 26
Total swap : 13

Merge Sort:

<table>
<tr><td colspan="10">9,8,7,6,5,4,3,2,1,0</td></tr>
<tr><td colspan="10">Size = 10
l(left) = 0
r(right) = size -1 = 9
l < r, divide array by two
m(middle) = floor((l+r)/2) = 4</td></tr>
<tr><td colspan="5">9,8,7,6,5</td><td colspan="5">4,3,2,1,0</td></tr>
<tr><td colspan="5">l = 0
r = 4
l < r
m = 2</td><td colspan="5">l = 5
r = 9
l < r
m = 7</td></tr>
<tr><td colspan="3">9,8,7</td><td colspan="2">6,5</td><td colspan="3">4,3,2</td><td colspan="2">1,0</td></tr>
<tr><td colspan="3">l = 0
r = 2
l < r
m = 1</td><td colspan="2">l = 3
r = 4
l < r
m = 3</td><td colspan="3">l = 5
r = 7
l < r
m = 6</td><td colspan="2">l = 8
r = 9
l < r
m = 8</td></tr>
<tr><td colspan="2">9,8</td><td>7</td><td>6</td><td>5</td><td colspan="2">4,3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="2">l = 0
r = 1
l < r
m = 0</td><td>l = 2
r = 2
l >= r</td><td>l = 2
r = 2
l >= r</td><td>l = 4
r = 4
l >= r</td><td colspan="2">l = 5
r = 6
l < r
m = 5</td><td>l = 7
r = 7
l >= r</td><td>l = 8
r = 8
l >= r</td><td>l = 9
r = 9
l >= r</td></tr>
<tr><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>l = 0
r = 0
l >= r</td><td>l = 1
r = 1
l >= r</td><td></td><td></td><td></td><td>l = 5
r = 5
l >= r</td><td>l = 6
r = 6
l >= r</td><td></td><td></td><td></td></tr>
<tr><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="2">9 < 8 ? false</td><td></td><td></td><td></td><td colspan="2">4 < 3 ? false</td><td></td><td></td><td></td></tr>
<tr><td colspan="2">8,9</td><td>7</td><td>6</td><td>5</td><td colspan="2">3,4</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="3">8 < 7 ? false</td><td colspan="2">6 < 5 ? false</td><td colspan="3">3 < 2 ? false</td><td colspan="2">1 < 0 ? false</td></tr>
<tr><td colspan="3">7,8,9</td><td colspan="2">5,6</td><td colspan="3">2,3,4</td><td colspan="2">0,1</td></tr>
<tr><td colspan="5">7 < 5 ? false
7 < 6 ? false</td><td colspan="5">2 < 0 ? false
2 < 1 ? false</td></tr>
</table>

5,6,7,8,9	0,1,2,3,4
5 < 0 ? false 5 < 1 ? false 5 < 2 ? false 5 < 3 ? false 5 < 4 ? false	
0,1,2,3,4,5,6,7,8,9	

Total comparision : 34 (base case + comprision between array elements)
Total displacement : 34 (replace elements a new array)

Heap Sort:

B = {9,8,7,6,5,4,3,2,1,0}				
Build a Max Heap array				
Algorithm: In the default array, the value corresponding to the array index is compared to the value corresponding to the parent index ((array index-1) / 2) in the max heap array. If the operation is true, their positions are swapped. This process continues until the comparision operation is false or has no parent. The general process continues until array index is smaller than size.				
Array Index	**Max Heap Array**	**Parent Index**	**Description**	**After operation Max Heap Array**
0	9,8,7,6,5,4,3,2,1,0	-	Initially 0. index is root	9,8,7,6,5,4,3,2,1,0
1	9,8,7,6,5,4,3,2,1,0	0	8 > 9 ? false	9,8,7,6,5,4,3,2,1,0
2	9,8,7,6,5,4,3,2,1,0	0	7 > 9 ? false	9,8,7,6,5,4,3,2,1,0
3	9,8,7,6,5,4,3,2,1,0	1	6 > 8 ? false	9,8,7,6,5,4,3,2,1,0
4	9,8,7,6,5,4,3,2,1,0	1	5 > 8 ? false	9,8,7,6,5,4,3,2,1,0
5	9,8,7,6,5,4,3,2,1,0	2	4 > 7 ? false	9,8,7,6,5,4,3,2,1,0
6	9,8,7,6,5,4,3,2,1,0	2	3 > 7 ? false	9,8,7,6,5,4,3,2,1,0
7	9,8,7,6,5,4,3,2,1,0	3	2 > 6 ? false	9,8,7,6,5,4,3,2,1,0
8	9,8,7,6,5,4,3,2,1,0	3	1 > 6 ? false	9,8,7,6,5,4,3,2,1,0
9	9,8,7,6,5,4,3,2,1,0	4	0 > 5 ? false	9,8,7,6,5,4,3,2,1,0
10			Max heap array is created	9,8,7,6,5,4,3,2,1,0

B = {9,8,7,6,5,4,3,2,1,0}			
Sorting			
Algorithm: Swap array index with head in array. Heapify is applied to the part up to the array index in the array. Sorting is done until the array index is 0.			
Array Index	**Array**	**Description**	**After operation array**
9	9,8,7,6,5,4,3,2,1,0	Head and array index were swapped. Apply heapify	0,8,7,6,5,4,3,2,1,9
	0,8,7,6,5,4,3,2,1,9	8 > 7 ? true max child = 8 0 < 8 ? true – swap them	8,0,7,6,5,4,3,2,1,9
	8,0,7,6,5,4,3,2,1,9	7 > 6 ? true max child = 7 0 < 7 ? true – swap them	8,7,0,6,5,4,3,2,1,9
	8,7,0,6,5,4,3,2,1,9	4 > 3 ? true max child = 4	8,7,4,6,5,0,3,2,1,9

		0 < 4 ? true – swap them	
	8,7,4,6,5,0,3,2,1,9	Element 0 doesn't have children	8,7,4,6,5,0,3,2,1,9
8	8,7,4,6,5,0,3,2,1,9	Head and array index were swapped. Apply heapify	1,7,4,6,5,0,3,2,8,9
	1,7,4,6,5,0,3,2,8,9	7 > 4 ? true max child = 7 1 < 7 ? true – swap them	7,1,4,6,5,0,3,2,8,9
	7,1,4,6,5,0,3,2,8,9	6 > 5 ? true max child = 6 1 < 6 ? true – swap them	7,6,4,1,5,0,3,2,8,9
	7,6,4,1,5,0,3,2,8,9	Max child = 2 1 < 2 ? true – swap them	7,6,4,2,5,0,3,1,8,9
	7,6,4,2,5,0,3,1,8,9	Element 1 doesn't have children	7,6,4,2,5,0,3,1,8,9
7	7,6,4,2,5,0,3,1,8,9	Head and array index were swapped. Apply heapify	1,6,4,2,5,0,3,7,8,9
	1,6,4,2,5,0,3,7,8,9	6 > 4 ? true max child = 6 1 < 6 ? true – swap them	6,1,4,2,5,0,3,7,8,9
	6,1,4,2,5,0,3,7,8,9	5 > 0 ? true max child = 5 1 < 5 ? true – swap them	6,5,4,2,1,0,3,7,8,9
	6,5,4,2,1,0,3,7,8,9	Element 1 doesn't have children	6,5,4,2,1,0,3,7,8,9
6	6,5,4,2,1,0,3,7,8,9	Head and array index were swapped. Apply heapify	3,5,4,2,1,0,6,7,8,9
	3,5,4,2,1,0,6,7,8,9	5 > 4 ? true max child = 5 3 < 5 ? true – swap them	5,3,4,2,1,0,6,7,8,9
	5,3,4,2,1,0,6,7,8,9	2 > 1 ? true max child = 2 3 < 2 false	5,3,4,2,1,0,6,7,8,9
5	5,3,4,2,1,0,6,7,8,9	Head and array index were swapped. Apply heapify	0,3,4,2,1,5,6,7,8,9
	0,3,4,2,1,5,6,7,8,9	3 > 4 ? false max child = 4 0 < 4 ? true – swap them	4,3,0,2,1,5,6,7,8,9
	4,3,0,2,1,5,6,7,8,9	Element 0 doesn't have children	4,3,0,2,1,5,6,7,8,9
4	4,3,0,2,1,5,6,7,8,9	Head and array index were swapped. Apply heapify	1,3,0,2,4,5,6,7,8,9
	1,3,0,2,4,5,6,7,8,9	3 > 0 ? true max child = 3 1 < 3 ? true – swap them	3,1,0,2,4,5,6,7,8,9
	3,1,0,2,4,5,6,7,8,9	Max child = 2 1 < 2 ? true – swap them	3,2,0,1,4,5,6,7,8,9
	3,2,0,1,4,5,6,7,8,9	Element 1 doesn't have children	3,2,0,1,4,5,6,7,8,9
3	3,2,0,1,4,5,6,7,8,9	Head and array index were swapped. Apply heapify	1,2,0,3,4,5,6,7,8,9
	1,2,0,3,4,5,6,7,8,9	2 > 0 ? true max child = 2 1 < 2 ? true – swap them	2,1,0,3,4,5,6,7,8,9
	2,1,0,3,4,5,6,7,8,9	Element 1 doesn't have children	2,1,0,3,4,5,6,7,8,9
2	2,1,0,3,4,5,6,7,8,9	Head and array index were swapped. Apply heapify	0,1,2,3,4,5,6,7,8,9
	0,1,2,3,4,5,6,7,8,9	Max child = 1 0 < 1 ? true – swap them	1,0,2,3,4,5,6,7,8,9
	1,0,2,3,4,5,6,7,8,9	Element 0 doesn't have children	1,0,2,3,4,5,6,7,8,9
1	1,0,2,3,4,5,6,7,8,9	Head and array index were swapped.	0,1,2,3,4,5,6,7,8,9
	0,1,2,3,4,5,6,7,8,9	Element 0 doesn't have children	0,1,2,3,4,5,6,7,8,9
0	0,1,2,3,4,5,6,7,8,9	Array is sorted	0,1,2,3,4,5,6,7,8,9

Total comparision : 47 (build + sort)
Total displacement : 23 (swap amount)

Quick Sort:

Pivot: Always picked last element as pivot
Partition Algorithm
The logic is simple, we start from the leftmost element and keep track of index of smaller elements as i. While traversing, if we find a smaller element, we swap current element with arr[i]. Otherwise we ignore current element.

B = {9,8,7,6,5,4,3,2,1,0}

Array	Pivot Value	p	i	end	Description	After operation array
9,8,7,6,5,4,3,2,1,0			0	9	i < end ? true – partition calls.	
9,8,7,6,5,4,3,2,1,0	0	-1	0	9	9 < 0 ? false	
9,8,7,6,5,4,3,2,1,0	0	-1	1	9	8 < 0 ? false	
9,8,7,6,5,4,3,2,1,0	0	-1	2	9	7 < 0 ? false	
9,8,7,6,5,4,3,2,1,0	0	-1	3	9	6 < 0 ? false	
9,8,7,6,5,4,3,2,1,0	0	-1	4	9	5 < 0 ? false	
9,8,7,6,5,4,3,2,1,0	0	-1	5	9	4 < 0 ? false	
9,8,7,6,5,4,3,2,1,0	0	-1	6	9	3 < 0 ? false	
9,8,7,6,5,4,3,2,1,0	0	-1	7	9	2 < 0 ? false	
9,8,7,6,5,4,3,2,1,0	0	-1	8	9	1 < 0 ? false	
9,8,7,6,5,4,3,2,1,0	0	-1	9	9	i < end ? false. Last operation ++p and swap p and pivot (9,0). Partition is finish. So next recursion calls Left part: i = 0, end(p-1) = -1 Right part: i(p+1) = 1, end = 9	0,8,7,6,5,4,3,2,1,9
0,8,7,6,5,4,3,2,1,9			0	-1	i < end ? false	
0,8,7,6,5,4,3,2,1,9			1	9	i < end ? true – partition calls.	
0,8,7,6,5,4,3,2,1,9	9	0	1	9	8 < 9 ? true, ++p, swap p and i (8,8)	0,8,7,6,5,4,3,2,1,9
0,8,7,6,5,4,3,2,1,9	9	1	2	9	7 < 9 ? true, ++p, swap p and i (7,7)	0,8,7,6,5,4,3,2,1,9
0,8,7,6,5,4,3,2,1,9	9	2	3	9	6 < 9 ? true, ++p, swap p and i (6,6)	0,8,7,6,5,4,3,2,1,9
0,8,7,6,5,4,3,2,1,9	9	3	4	9	7 < 9 ? true, ++p, swap p and i (5,5)	0,8,7,6,5,4,3,2,1,9
0,8,7,6,5,4,3,2,1,9	9	4	5	9	4 < 9 ? true, ++p, swap p and i (4,4)	0,8,7,6,5,4,3,2,1,9
0,8,7,6,5,4,3,2,1,9	9	5	6	9	3 < 9 ? true, ++p, swap p and i (3,3)	0,8,7,6,5,4,3,2,1,9
0,8,7,6,5,4,3,2,1,9	9	6	7	9	2 < 9 ? true, ++p, swap p and i (2,2)	0,8,7,6,5,4,3,2,1,9
0,8,7,6,5,4,3,2,1,9	9	7	8	9	1 < 9 ? true, ++p, swap p and i (1,1)	0,8,7,6,5,4,3,2,1,9
0,8,7,6,5,4,3,2,1,9	9	8	9	9	i < end ? false. Last operation ++p and swap p and pivot (9,9). Partition is finish. So next recursion calls Left part: i = 1, end(p-1) = 8 Right part: i(p+1) = 10, end = 9	
0,8,7,6,5,4,3,2,1,9			1	8	i < end ? true – partition calls.	
0,8,7,6,5,4,3,2,1,9	1	0	1	8	8 < 1 ? false	
0,8,7,6,5,4,3,2,1,9	1	0	2	8	7 < 1 ? false	
0,8,7,6,5,4,3,2,1,9	1	0	3	8	6 < 1 ? false	
0,8,7,6,5,4,3,2,1,9	1	0	4	8	5 < 1 ? false	

0,8,7,6,5,4,3,2,1,9	1	0	5	8	4 < 1 ? false	
0,8,7,6,5,4,3,2,1,9	1	0	6	8	3 < 1 ? false	
0,8,7,6,5,4,3,2,1,9	1	0	7	8	2 < 1 ? false	
0,8,7,6,5,4,3,2,1,9	1	0	8	8	i < end ? false. Last operation ++p and swap p and pivot (8,1). Partition is finish. So next recursion calls Left part: i = 1, end(p-1) = 0 Right part: i(p+1) = 2, end = 8	0,1,7,6,5,4,3,2,8,9
0,1,7,6,5,4,3,2,8,9			1	0	i < end ? false	
0,1,7,6,5,4,3,2,8,9			2	8	i < end ? true – partition calls.	
0,1,7,6,5,4,3,2,8,9	8	1	2	8	7 < 8 ? true, ++p , swap p and i (7,7)	0,1,7,6,5,4,3,2,8,9
0,1,7,6,5,4,3,2,8,9	8	2	3	8	6 < 8 ? true, ++p , swap p and i (6,6)	0,1,7,6,5,4,3,2,8,9
0,1,7,6,5,4,3,2,8,9	8	3	4	8	5 < 8 ? true, ++p , swap p and i (5,5)	0,1,7,6,5,4,3,2,8,9
0,1,7,6,5,4,3,2,8,9	8	4	5	8	4 < 8 ? true, ++p , swap p and i (4,4)	0,1,7,6,5,4,3,2,8,9
0,1,7,6,5,4,3,2,8,9	8	5	6	8	3 < 8 ? true, ++p , swap p and i (3,3)	0,1,7,6,5,4,3,2,8,9
0,1,7,6,5,4,3,2,8,9	8	6	7	8	2 < 8 ? true, ++p , swap p and i (2,2)	0,1,7,6,5,4,3,2,8,9
0,1,7,6,5,4,3,2,8,9	8	7	8	8	i < end ? false. Last operation ++p and swap p and pivot (8,8). Partition is finish. So next recursion calls Left part: i = 2, end(p-1) = 7 Right part: i(p+1) = 9, end = 8	0,1,7,6,5,4,3,2,8,9
0,1,7,6,5,4,3,2,8,9			2	7	i < end ? true – partition calls.	
0,1,7,6,5,4,3,2,8,9	2	1	2	7	7 < 2 ? false	
0,1,7,6,5,4,3,2,8,9	2	1	3	7	6 < 2 ? false	
0,1,7,6,5,4,3,2,8,9	2	1	4	7	5 < 2 ? false	
0,1,7,6,5,4,3,2,8,9	2	1	5	7	4 < 2 ? false	
0,1,7,6,5,4,3,2,8,9	2	1	6	7	3 < 2 ? false	
0,1,7,6,5,4,3,2,8,9	2	1	7	7	i < end ? false. Last operation ++p and swap p and pivot (7,2). Partition is finish. So next recursion calls Left part: i = 2, end(p-1) = 1 Right part: i(p+1) = 3, end = 7	0,1,2,6,5,4,3,7,8,9
0,1,2,6,5,4,3,7,8,9			2	1	i < end ? false	
0,1,2,6,5,4,3,7,8,9			3	7	i < end ? true – partition calls.	
0,1,2,6,5,4,3,7,8,9	7	2	3	7	6 < 7 ? true, ++p , swap p and i (6,6)	0,1,2,6,5,4,3,7,8,9
0,1,2,6,5,4,3,7,8,9	7	3	4	7	5 < 7 ? true, ++p , swap p and i (5,5)	0,1,2,6,5,4,3,7,8,9
0,1,2,6,5,4,3,7,8,9	7	4	5	7	4 < 7 ? true, ++p , swap p and i (4,4)	0,1,2,6,5,4,3,7,8,9
0,1,2,6,5,4,3,7,8,9	7	5	6	7	3 < 7 ? true, ++p , swap p and i (3,3)	0,1,2,6,5,4,3,7,8,9
0,1,2,6,5,4,3,7,8,9	7	6	7	7	i < end ? false. Last operation ++p and swap p and pivot (7,7). Partition is finish. So next recursion calls Left part: i = 3, end(p-1) = 6 Right part: i(p+1) = 8, end = 7	0,1,2,6,5,4,3,7,8,9
0,1,2,6,5,4,3,7,8,9			3	6	i < end ? true – partition calls.	
0,1,2,6,5,4,3,7,8,9	3	2	3	6	6 < 3 ? false	
0,1,2,6,5,4,3,7,8,9	3	2	4	6	5 < 3 ? false	
0,1,2,6,5,4,3,7,8,9	3	2	5	6	4 < 3 ? false	
0,1,2,6,5,4,3,7,8,9	3	2	6	6	i < end ? false. Last operation ++p and swap p and pivot (6,3). Partition is finish. So next recursion calls	0,1,2,3,5,4,6,7,8,9

					Left part: i = 3, end(p-1) = 2 Right part: i(p+1) = 4, end = 6	
0,1,2,3,5,4,6,7,8,9			3	2	i < end ? false	
0,1,2,3,5,4,6,7,8,9			4	6	i < end ? true – partition calls.	
0,1,2,3,5,4,6,7,8,9	6	3	4	6	5 < 6 ? true, ++p , swap p and i (5,5)	0,1,2,3,5,4,6,7,8,9
0,1,2,3,5,4,6,7,8,9	6	4	5	6	4 < 6 ? true, ++p , swap p and i (4,4)	0,1,2,3,5,4,6,7,8,9
0,1,2,3,5,4,6,7,8,9	6	5	6	6	i < end ? false. Last operation ++p and swap p and pivot (6,6). Partition is finish. So next recursion calls Left part: i = 4, end(p-1) = 5 Right part: i(p+1) = 7, end = 6	0,1,2,3,5,4,6,7,8,9
0,1,2,3,5,4,6,7,8,9			4	5	i < end ? true – partition calls.	
0,1,2,3,5,4,6,7,8,9	4	3	4	5	5 < 4 ? false	
0,1,2,3,5,4,6,7,8,9	4	3	5	5	i < end ? false. Last operation ++p and swap p and pivot (5,4). Partition is finish. So next recursion calls Left part: i = 4, end(p-1) = 3 Right part: i(p+1) = 6, end = 5	0,1,2,3,4,5,6,7,8,9
0,1,2,3,4,5,6,7,8,9			4	3	i < end ? false	
0,1,2,3,4,5,6,7,8,9			6	5	i < end ? false	
0,1,2,3,4,5,6,7,8,9			7	6	i < end ? false	
0,1,2,3,4,5,6,7,8,9			8	7	i < end ? false	
0,1,2,3,4,5,6,7,8,9			9	8	i < end ? false	
0,1,2,3,4,5,6,7,8,9			10	9	i < end ? false	
					Array is sorted	0,1,2,3,4,5,6,7,8,9

Total comparision : 73 (base case + comparision between array elements)
Total displacement : 29 (swap amount)

- C = {5,2,13,9,1,7,6,8,1,15,4,11}

Shell Sort:

pass = 1
gap = floor(size/2) = floor(12/2) = floor(6) = 6

It is made a **virtual** sub-list of all values located at the interval of gap positions.
{5,6},{2,8},{13,1},{9,15},{1,4},{7,11}

It is compared values in each sub-list and swap them (if necessary) in the original array.
5 > 6 ? false
2 > 8 ? false
13 > 1 ? true - swap them C = {5,2,1,9,1,7,6,8,13,15,4,11}
9 > 15 ? false
1 > 4 ? false
7 > 11 ? false

Then calculate gap again.
pass = 2

gap = floor(gap/2) = floor(6/2) = floor(3) = 3

C = {5,2,1,9,1,7,6,8,13,15,4,11}
It is made a **virtual** sub-list of all values located at the interval of gap positions.
{5,9,6,15},{2,1,8,4},{1,7,13,11}

It is compared values in each sub-list and swap them (if necessary) in the original array.
5 > 9 ? false
2 > 1 ? true - swap them C = {5,1,1,9,2,7,6,8,13,15,4,11}
1 > 7 ? false
9 > 6 ? true - swap them C = {5,1,1,6,2,7,9,8,13,15,4,11}
5 > 6 ? false
2 > 8 ? false
7 > 13 ? false
9 > 15 ? false
8 > 4 ? true - swap them C = {5,1,1,6,2,7,9,4,13,15,8,11}
2 > 4 ? false
13 > 11 ? true - swap them C = {5,1,1,6,2,7,9,4,11,15,8,13}
7 > 11 ? false

Then calculate gap again.
pass = 3
gap = floor(gap/2) = floor(3/2) = floor(1,5) = 1

C = {5,1,1,6,2,7,9,4,11,15,8,13}
Finally, we sort the rest of the array using interval of value 1. Shell sort uses insertion sort to sort the array.

5 > 1 ? true - swap them C = {1,5,1,6,2,7,9,4,11,15,8,13}
5 > 1 ? true - swap them C = {1,1,5,6,2,7,9,4,11,15,8,13}
1 > 1 ? false
5 > 6 ? false
6 > 2 ? true - swap them C = {1,1,5,2,6,7,9,4,11,15,8,13}
5 > 2 ? true - swap them C = {1,1,2,5,6,7,9,4,11,15,8,13}
1 > 2 ? false
6 > 7 ? false
7 > 9 ? false
9 > 4 ? true - swap them C = {1,1,2,5,6,7,4,9,11,15,8,13}
7 > 4 ? true - swap them C = {1,1,2,5,6,4,7,9,11,15,8,13}
6 > 4 ? true - swap them C = {1,1,2,5,4,6,7,9,11,15,8,13}
5 > 4 ? true - swap them C = {1,1,2,4,5,6,7,9,11,15,8,13}
2 > 4 ? false
9 > 11 ? false
11 > 15 ? false
15 > 8 ? true - swap them C = {1,1,2,4,5,6,7,9,11,8,15,13}
11 > 8 ? true - swap them C = {1,1,2,4,5,6,7,9,8,11,15,13}
9 > 8 ? true - swap them C = {1,1,2,4,5,6,7,8,9,11,15,13}
7 > 8 ? false

15 > 13 ? true - swap them C = {1,1,2,4,5,6,7,8,9,11,13,15}
11 > 13 ? false
Then calculate gap again
gap = floor(gap/2) = floor(1/2) = floor(0,5) = 0
And shell sort algroitm is over. Array was sorted.

Size : 12
Total comparision : 40
Total swap : 17

Merge Sort:

<table>
<tr><td colspan="12">5,2,13,9,1,7,6,8,1,15,4,11</td></tr>
<tr><td colspan="12">Size = 12
l(left) = 0
r(right) = size -1 = 11
l < r, divide array by two
m(middle) = floor((l+r)/2) = 5</td></tr>
<tr><td colspan="6">5,2,13,9,1,7</td><td colspan="6">6,8,1,15,4,11</td></tr>
<tr><td colspan="6">l = 0
r = 5
l < r
m = 2</td><td colspan="6">l = 6
r = 11
l < r
m = 8</td></tr>
<tr><td colspan="3">5,2,13</td><td colspan="3">9,1,7</td><td colspan="3">6,8,1</td><td colspan="3">15,4,11</td></tr>
<tr><td colspan="3">l = 0
r = 2
l < r
m = 1</td><td colspan="3">l = 3
r = 5
l < r
m = 4</td><td colspan="3">l = 6
r = 8
l < r
m = 7</td><td colspan="3">l = 9
r = 11
l < r
m = 10</td></tr>
<tr><td colspan="2">5,2</td><td>13</td><td colspan="2">9,1</td><td>7</td><td colspan="2">6,8</td><td>1</td><td colspan="2">15,4</td><td>11</td></tr>
<tr><td colspan="2">l = 0
r = 1
l < r
m = 0</td><td>l = 2
r = 2
l >=r</td><td colspan="2">l = 3
r = 4
l < r
m = 3</td><td>l = 5
r = 5
l >=r</td><td colspan="2">l = 6
r = 7
l < r
m = 6</td><td>l = 8
r = 8
l >=r</td><td colspan="2">l = 9
r = 10
l < r
m = 9</td><td>l =11
r =11
l >=r</td></tr>
<tr><td>5</td><td>2</td><td>13</td><td>9</td><td>1</td><td>7</td><td>6</td><td>8</td><td>1</td><td>15</td><td>4</td><td>11</td></tr>
<tr><td>l = 0
r = 0
l >=r</td><td>l = 1
r = 1
l >=r</td><td></td><td>l = 3
r = 3
l >=r</td><td>l = 4
r = 4
l >=r</td><td></td><td>l = 6
r = 6
l >=r</td><td>l = 7
r = 7
l >=r</td><td></td><td>l = 9
r = 9
l >=r</td><td>l =10
r =10
l >=r</td><td></td></tr>
<tr><td>5</td><td>2</td><td>13</td><td>9</td><td>1</td><td>7</td><td>6</td><td>8</td><td>1</td><td>15</td><td>4</td><td>11</td></tr>
<tr><td colspan="2">5 < 2 ? false</td><td></td><td colspan="2">9 < 1 ? false</td><td></td><td colspan="2">6 < 8 ? true</td><td></td><td colspan="2">15< 4 ? false</td><td></td></tr>
<tr><td colspan="2">2,5</td><td>13</td><td colspan="2">1,9</td><td>7</td><td colspan="2">6,8</td><td>1</td><td colspan="2">4,15</td><td>11</td></tr>
<tr><td colspan="3">2 < 13 ? true
5 < 13 ? true</td><td colspan="3">1 < 7 ? true
9 < 7 ? false</td><td colspan="3">6 < 1 ? false</td><td colspan="3">4 < 11 ? true
15 < 11 ? false</td></tr>
<tr><td colspan="3">2,5,13</td><td colspan="3">1,7,9</td><td colspan="3">1,6,8</td><td colspan="3">4,11,15</td></tr>
<tr><td colspan="6">2 < 1 ? false
2 < 7 ? true
5 < 7 ? true
13 < 7 ? false
13 < 9 ? false</td><td colspan="6">1 < 4 ? true
6 < 4 ? false
6 < 11 ? true
8 < 11 ? true</td></tr>
<tr><td colspan="6">1,2,5,7,9,13</td><td colspan="6">1,4,6,8,11,15</td></tr>
<tr><td colspan="12">1 < 1 ? false</td></tr>
</table>

1 < 4 ? true 2 < 4 ? true 5 < 4 ? false 5 < 6 ? true 7 < 6 ? false 7 < 8 ? true 9 < 8 ? false 9 < 11 ? true 13 < 11 ? false 13 < 15 ? true
1,1,2,4,5,6,7,8,9,11,13,15

Total comparision : 54 (base case + comprision between array elements)
Total displacement : 44 (replace elements a new array)

Heap Sort:

C = {5,2,13,9,1,7,6,8,1,15,4,11}				
Build a Max Heap array				
Algorithm: In the default array, the value corresponding to the array index is compared to the value corresponding to the parent index ((array index-1) / 2) in the max heap array. If the operation is true, their positions are swapped. This process continues until the comparision operation is false or has no parent. The general process continues until array index is smaller than size.				
Array Index	**Max Heap Array**	**Parent Index**	**Description**	**After operation Max Heap Array**
0	5,2,13,9,1,7,6,8,1,15,4,11	-	Initially 0. index is root	5,2,13,9,1,7,6,8,1,15,4,11
1	5,2,13,9,1,7,6,8,1,15,4,11	0	2 > 5 ? false	5,2,13,9,1,7,6,8,1,15,4,11
2	5,2,13,9,1,7,6,8,1,15,4,11	0	13 > 5 ? true – swap them	13,2,5,9,1,7,6,8,1,15,4,11
2	13,2,5,9,1,7,6,8,1,15,4,11	-	Element 13 doesn't have parent	13,2,5,9,1,7,6,8,1,15,4,11
3	13,2,5,9,1,7,6,8,1,15,4,11	1	9 > 2 ? true – swap them	13,9,5,2,1,7,6,8,1,15,4,11
3	13,9,5,2,1,7,6,8,1,15,4,11	0	9 > 13 ? false	13,9,5,2,1,7,6,8,1,15,4,11
4	13,9,5,2,1,7,6,8,1,15,4,11	1	1 > 9 ? false	13,9,5,2,1,7,6,8,1,15,4,11
5	13,9,5,2,1,7,6,8,1,15,4,11	2	7 > 5 ? true – swap them	13,9,7,2,1,5,6,8,1,15,4,11
5	13,9,7,2,1,5,6,8,1,15,4,11	0	7 > 13 ? false	13,9,7,2,1,5,6,8,1,15,4,11
6	13,9,7,2,1,5,6,8,1,15,4,11	2	6 > 7 ? false	13,9,7,2,1,5,6,8,1,15,4,11
7	13,9,7,2,1,5,6,8,1,15,4,11	3	8 > 2 ? true – swap them	13,9,7,8,1,5,6,2,1,15,4,11
7	13,9,7,8,1,5,6,2,1,15,4,11	1	8 > 9 ? false	13,9,7,8,1,5,6,2,1,15,4,11
8	13,9,7,8,1,5,6,2,1,15,4,11	3	1 > 8 ? false	13,9,7,8,1,5,6,2,1,15,4,11
9	13,9,7,8,1,5,6,2,1,15,4,11	4	15 > 1 ? true – swap them	13,9,7,8,15,5,6,2,1,1,4,11
9	13,9,7,8,15,5,6,2,1,1,4,11	1	15 > 9 ? true – swap them	13,15,7,8,9,5,6,2,1,1,4,11
9	13,15,7,8,9,5,6,2,1,1,4,11	0	15 > 13 ? true – swap them	15,13,7,8,9,5,6,2,1,1,4,11
9	15,13,7,8,9,5,6,2,1,1,4,11	-	Element 15 doesn't have parent	15,13,7,8,9,5,6,2,1,1,4,11
10	15,13,7,8,9,5,6,2,1,1,4,11	4	4 > 9 ? false	15,13,7,8,9,5,6,2,1,1,4,11
11	15,13,7,8,9,5,6,2,1,1,4,11	5	11 > 5 ? true – swap them	15,13,7,8,9,11,6,2,1,1,4,5
11	15,13,7,8,9,11,6,2,1,1,4,5	2	11 > 7 ? true – swap them	15,13,11,8,9,7,6,2,1,1,4,5
11	15,13,11,8,9,7,6,2,1,1,4,5	0	11 > 15 ? false	15,13,11,8,9,7,6,2,1,1,4,5
12			Max heap array is created	15,13,11,8,9,7,6,2,1,1,4,5

C = {15,13,11,8,9,7,6,2,1,1,4,5}			
Sorting			
Algorithm: Swap array index with head in array. Heapify is applied to the part up to the array index in the array. Sorting is done until the array index is 0.			
Array Index	**Array**	**Description**	**After operation array**
11	15,13,11,8,9,7,6,2,1,1,4,5	Head and array index were swapped. Apply heapify	5,13,11,8,9,7,6,2,1,1,4,15
	5,13,11,8,9,7,6,2,1,1,4,15	13 > 11 ? true max child = 13 5 < 13 ? true – swap them	13,5,11,8,9,7,6,2,1,1,4,15
	13,5,11,8,9,7,6,2,1,1,4,15	8 > 9 ? false max child = 9 5 < 9 ? true – swap them	13,9,11,8,5,7,6,2,1,1,4,15
	13,9,11,8,5,7,6,2,1,1,4,15	1 > 4 ? false max child = 4 5 < 4 ? false	13,9,11,8,5,7,6,2,1,1,4,15
10	13,9,11,8,5,7,6,2,1,1,4,15	Head and array index were swapped. Apply heapify	4,9,11,8,5,7,6,2,1,1,13,15
	4,9,11,8,5,7,6,2,1,1,13,15	9 > 11 ? false max child = 11 4 < 11 ? true – swap them	11,9,4,8,5,7,6,2,1,1,13,15
	11,9,4,8,5,7,6,2,1,1,13,15	7 > 6 ? true max child = 7 4 < 7 ? true – swap them	11,9,7,8,5,4,6,2,1,1,13,15
	11,9,7,8,5,4,6,2,1,1,13,15	Element 4 doesn't have children	11,9,7,8,5,4,6,2,1,1,13,15
9	11,9,7,8,5,4,6,2,1,1,13,15	Head and array index were swapped. Apply heapify	1,9,7,8,5,4,6,2,1,11,13,15
	1,9,7,8,5,4,6,2,1,11,13,15	9 > 7 ? true max child = 9 1 < 9 ? true – swap them	9,1,7,8,5,4,6,2,1,11,13,15
	9,1,7,8,5,4,6,2,1,11,13,15	8 > 5 ? true max child = 8 1 < 8 ? true – swap them	9,8,7,1,5,4,6,2,1,11,13,15
	9,8,7,1,5,4,6,2,1,11,13,15	2 > 1 ? true max child = 2 1 < 2 ? true – swap them	9,8,7,2,5,4,6,1,1,11,13,15
	9,8,7,2,5,4,6,1,1,11,13,15	Element 1 doesn't have children	9,8,7,2,5,4,6,1,1,11,13,15
8	9,8,7,2,5,4,6,1,1,11,13,15	Head and array index were swapped. Apply heapify	1,8,7,2,5,4,6,1,9,11,13,15
	1,8,7,2,5,4,6,1,9,11,13,15	8 > 7 ? true max child = 8 1< 8 ? true – swap them	8,1,7,2,5,4,6,1,9,11,13,15
	8,1,7,2,5,4,6,1,9,11,13,15	2 > 5 ? false max child = 5 1 < 5 true – swap them	8,5,7,2,1,4,6,1,9,11,13,15
	8,5,7,2,1,4,6,1,9,11,13,15	Element 1 doesn't have children	8,5,7,2,1,4,6,1,9,11,13,15
7	8,5,7,2,1,4,6,1,9,11,13,15	Head and array index were swapped. Apply heapify	1,5,7,2,1,4,6,8,9,11,13,15
	1,5,7,2,1,4,6,8,9,11,13,15	5 > 7 ? false max child = 7 1 < 7 ? true – swap them	7,5,1,2,1,4,6,8,9,11,13,15
	7,5,1,2,1,4,6,8,9,11,13,15	4 > 6 ? false max child = 6 1 < 6 ? true – swap them	7,5,6,2,1,4,1,8,9,11,13,15
	7,5,6,2,1,4,1,8,9,11,13,15	Element 1 doesn't have children	7,5,6,2,1,4,1,8,9,11,13,15
6	7,5,6,2,1,4,1,8,9,11,13,15	Head and array index were swapped. Apply heapify	1,5,6,2,1,4,7,8,9,11,13,15
	1,5,6,2,1,4,7,8,9,11,13,15	5 > 6 ? false max child = 6 1 < 6 ? true – swap them	6,5,1,2,1,4,7,8,9,11,13,15
	6,5,1,2,1,4,7,8,9,11,13,15	Max child = 4	6,5,4,2,1,1,7,8,9,11,13,15

		1 < 4 ? true – swap them	
	6,5,4,2,1,1,7,8,9,11,13,15	Element 1 doesn't have children	6,5,4,2,1,1,7,8,9,11,13,15
5	6,5,4,2,1,1,7,8,9,11,13,15	Head and array index were swapped. Apply heapify	1,5,4,2,1,6,7,8,9,11,13,15
	1,5,4,2,1,6,7,8,9,11,13,15	5 > 4 ? true max child = 5 1 < 5 ? true – swap them	5,1,4,2,1,6,7,8,9,11,13,15
	5,1,4,2,1,6,7,8,9,11,13,15	2 > 1 ? true max child = 2 1 < 2 ? true – swap them	5,2,4,1,1,6,7,8,9,11,13,15
	5,2,4,1,1,6,7,8,9,11,13,15	Element 1 doesn't have children	5,2,4,1,1,6,7,8,9,11,13,15
4	5,2,4,1,1,6,7,8,9,11,13,15	Head and array index were swapped. Apply heapify	1,2,4,1,5,6,7,8,9,11,13,15
	1,2,4,1,5,6,7,8,9,11,13,15	2 > 4 ? false max child = 4 1 < 4 ? true – swap them	4,2,1,1,5,6,7,8,9,11,13,15
	4,2,1,1,5,6,7,8,9,11,13,15	Element 1 doesn't have children	4,2,1,1,5,6,7,8,9,11,13,15
3	4,2,1,1,5,6,7,8,9,11,13,15	Head and array index were swapped.	1,2,1,4,5,6,7,8,9,11,13,15
	1,2,1,4,5,6,7,8,9,11,13,15	2 > 1 ? true max child = 2 1 < 2 ? true – swap them	2,1,1,4,5,6,7,8,9,11,13,15
	2,1,1,4,5,6,7,8,9,11,13,15	Element 1 doesn't have children	2,1,1,4,5,6,7,8,9,11,13,15
2	2,1,1,4,5,6,7,8,9,11,13,15	Head and array index were swapped.	1,1,2,4,5,6,7,8,9,11,13,15
	1,1,2,4,5,6,7,8,9,11,13,15	Max child = 1 1 < 1 ? false	1,1,2,4,5,6,7,8,9,11,13,15
1	1,1,2,4,5,6,7,8,9,11,13,15	Head and array index were swapped.	1,1,2,4,5,6,7,8,9,11,13,15
	1,1,2,4,5,6,7,8,9,11,13,15	Element 1 doesn't have children	1,1,2,4,5,6,7,8,9,11,13,15
0		Array is sorted	1,1,2,4,5,6,7,8,9,11,13,15

Total comparision : 67 (build + sort)
Total displacement : 37 (swap amount)

Quick Sort:

Pivot: Always picked last element as pivot
Partition Algorithm
The logic is simple, we start from the leftmost element and keep track of index of smaller elements as i. While traversing, if we find a smaller element, we swap current element with arr[i]. Otherwise we ignore current element.

Array	Pivot Value	p	i	end	Description	After operation array
5,2,13,9,1,7,6,8,1,15,4,11			0	11	i < end ? true – partition calls.	
5,2,13,9,1,7,6,8,1,15,4,11	11	-1	0	11	5 < 11 ? true, ++p, swap p and i (5,5)	5,2,13,9,1,7,6,8,1,15,4,11
5,2,13,9,1,7,6,8,1,15,4,11	11	0	1	11	2 < 11 ? true, ++p, swap p and i (2,2)	5,2,13,9,1,7,6,8,1,15,4,11
5,2,13,9,1,7,6,8,1,15,4,11	11	1	2	11	13 < 11 ? false	
5,2,13,9,1,7,6,8,1,15,4,11	11	1	3	11	9 < 11 ? true, ++p, swap p and i (13,9)	5,2,9,13,1,7,6,8,1,15,4,11
5,2,9,13,1,7,6,8,1,15,4,11	11	2	4	11	1 < 11 ? true, ++p, swap p and i (13,1)	5,2,9,1,13,7,6,8,1,15,4,11
5,2,9,1,13,7,6,8,1,15,4,11	11	3	5	11	7 < 11 ? true, ++p, swap p and i (13,7)	5,2,9,1,7,13,6,8,1,15,4,11
5,2,9,1,7,13,6,8,1,15,4,11	11	4	6	11	6 < 11 ? true, ++p, swap p and i (13,6)	5,2,9,1,7,6,13,8,1,15,4,11
5,2,9,1,7,6,13,8,1,15,4,11	11	5	7	11	8 < 11 ? true, ++p, swap p and i (13,8)	5,2,9,1,7,6,8,13,1,15,4,11
5,2,9,1,7,6,8,13,1,15,4,11	11	6	8	11	1 < 11 ? true, ++p, swap p and i (13,1)	5,2,9,1,7,6,8,1,13,15,4,11
5,2,9,1,7,6,8,1,13,15,4,11	11	7	9	11	15 < 11 ? false	

5,2,9,1,7,6,8,1,13,15,4,11	11	7	10	11	4 < 11 ? true, ++p, swap p and i (13,4)	5,2,9,1,7,6,8,1,4,15,13,11
5,2,9,1,7,6,8,1,4,15,13,11	11	8	11	11	i < end ? false. Last operation ++p and swap p and pivot (15,11). Partition is finish. So next recursion calls Left part: i = 0, end(p-1) = 8 Right part: i(p+1), = 10, end = 11	5,2,9,1,7,6,8,1,4,11,13,15
5,2,9,1,7,6,8,1,4,11,13,15			0	8	i < end ? true – partition calls	
5,2,9,1,7,6,8,1,4,11,13,15	4	-1	0	8	5 < 4 ? false	
5,2,9,1,7,6,8,1,4,11,13,15	4	-1	1	8	2 < 4 ? true, ++p, swap p and i (5,2)	2,5,9,1,7,6,8,1,4,11,13,15
2,5,9,1,7,6,8,1,4,11,13,15	4	0	2	8	9 < 4 ? false	
2,5,9,1,7,6,8,1,4,11,13,15	4	0	3	8	1 < 4 ? true, ++p, swap p and i (5,1)	2,1,9,5,7,6,8,1,4,11,13,15
2,1,9,5,7,6,8,1,4,11,13,15	4	1	4	8	7 < 4 ? false	
2,1,9,5,7,6,8,1,4,11,13,15	4	1	5	8	6 < 4 ? false	
2,1,9,5,7,6,8,1,4,11,13,15	4	1	6	8	8 < 4 ? false	
2,1,9,5,7,6,8,1,4,11,13,15	4	1	7	8	1 < 4 ? true, ++p, swap p and i (1,9)	2,1,1,5,7,6,8,9,4,11,13,15
2,1,1,5,7,6,8,9,4,11,13,15	4	2	8	8	i < end ? false. Last operation ++p and swap p and pivot (5,4). Partition is finish. So next recursion calls Left part: i = 0, end(p-1) = 2 Right part: i(p+1) = 4, end = 8	2,1,1,4,7,6,8,9,5,11,13,15
2,1,1,4,7,6,8,9,5,11,13,15			0	2	i < end ? true – partition calls	
2,1,1,4,7,6,8,9,5,11,13,15	1	-1	0	2	2 < 1 ? false	
2,1,1,4,7,6,8,9,5,11,13,15	1	-1	1	2	1 < 1 ? false	
2,1,1,4,7,6,8,9,5,11,13,15	1	-1	2	2	i < end ? false. Last operation ++p and swap p and pivot (2,1). Partition is finish. So next recursion calls Left part: i = 0, end(p-1) = -1 Right part: i(p+1) = 1, end = 2	1,1,2,4,7,6,8,9,5,11,13,15
1,1,2,4,7,6,8,9,5,11,13,15			0	-1	i < end ? false	
1,1,2,4,7,6,8,9,5,11,13,15			1	2	i < end ? true – partition calls	
1,1,2,4,7,6,8,9,5,11,13,15	2	0	1	2	1 < 2 ? true, ++p swap p and i (1,1)	1,1,2,4,7,6,8,9,5,11,13,15
1,1,2,4,7,6,8,9,5,11,13,15	2	1	2	2	i < end ? false. Last operation ++p and swap p and pivot (2,2). Partition is finish. So next recursion calls Left part: i = 1, end(p-1) = 1 Right part: i(p+1) = 3, end = 2	1,1,2,4,7,6,8,9,5,11,13,15
1,1,2,4,7,6,8,9,5,11,13,15			1	1	i < end ? false	
1,1,2,4,7,6,8,9,5,11,13,15			3	2	i < end ? false	
1,1,2,4,7,6,8,9,5,11,13,15			4	8	i < end ? true – partition calls	
1,1,2,4,7,6,8,9,5,11,13,15	5	3	4	8	7 < 5 ? false	
1,1,2,4,7,6,8,9,5,11,13,15	5	3	5	8	6 < 5 ? false	
1,1,2,4,7,6,8,9,5,11,13,15	5	3	6	8	8 < 5 ? false	
1,1,2,4,7,6,8,9,5,11,13,15	5	3	7	8	9 < 5 ? false	
1,1,2,4,7,6,8,9,5,11,13,15	5	3	8	8	i < end ? false. Last operation ++p and swap p and pivot (7,5). Partition is finish. So next recursion calls Left part: i = 4, end(p-1) = 3 Right part: i(p+1) = 5, end = 8	1,1,2,4,5,6,8,9,7,11,13,15
1,1,2,4,5,6,8,9,7,11,13,15			4	3	i < end ? false	
1,1,2,4,5,6,8,9,7,11,13,15			5	8	i < end ? true – partition calls	

1,1,2,4,5,6,8,9,7,11,13,15	7	4	5	8	6 < 7 ? true, ++p, swap p and i (6,6)	1,1,2,4,5,6,8,9,7,11,13,15
1,1,2,4,5,6,8,9,7,11,13,15	7	5	6	8	8 < 7 ? false	
1,1,2,4,5,6,8,9,7,11,13,15	7	5	7	8	9 < 7 ? false	
1,1,2,4,5,6,8,9,7,11,13,15	7	5	8	8	i < end ? false. Last operation ++p and swap p and pivot (8,7). Partition is finish. So next recursion calls Left part: i = 4, end(p-1) = 6 Right part: i(p+1) = 7, end = 8	1,1,2,4,5,6,7,9,8,11,13,15
1,1,2,4,5,6,7,9,8,11,13,15			5	5	i < end ? false	
1,1,2,4,5,6,7,9,8,11,13,15			7	8	i < end ? true – partition calls	
1,1,2,4,5,6,7,9,8,11,13,15	8	6	7	8	9 < 8 ? false	
1,1,2,4,5,6,7,9,8,11,13,15	8	6	8	8	i < end ? false. Last operation ++p and swap p and pivot (9,8). Partition is finish. So next recursion calls Left part: i = 7, end(p-1) = 6 Right part: i(p+1) = 8, end = 8	1,1,2,4,5,6,7,8,9,11,13,15
1,1,2,4,5,6,7,8,9,11,13,15			7	6	i < end ? false	
1,1,2,4,5,6,7,8,9,11,13,15			8	8	i < end ? false	
1,1,2,4,5,6,7,8,9,11,13,15			10	11	i < end ? true – partition calls	
1,1,2,4,5,6,7,8,9,11,13,15	15	9	10	11	13 < 15 ? true, ++p, swap p and i (13,13)	1,1,2,4,5,6,7,8,9,11,13,15
1,1,2,4,5,6,7,8,9,11,13,15	15	10	11	11	i < end ? false. Last operation ++p and swap p and pivot (15,15). Partition is finish. So next recursion calls Left part: i = 10, end(p-1) = 10 Right part: i(p+1) = 12, end = 11	1,1,2,4,5,6,7,8,9,11,13,15
1,1,2,4,5,6,7,8,9,11,13,15			10	10	i < end ? false	
1,1,2,4,5,6,7,8,9,11,13,15			12	11	İ < end ? false	
					Array is sorted	1,1,2,4,5,6,7,8,9,11,13,15

Total comparision : 56 (base case + comparision between array elements)
Total displacement : 23 (swap amount)

- D = {'S','B','I','M','H','Q','C','L','R','E','P','K'}

Shell Sort:

pass = 1
gap = floor(size/2) = floor(12/2) = floor(6) = 6

It is made a **virtual** sub-list of all values located at the interval of gap positions.
{'S','C'},{'B','L'},{'I','R'},{'M','E'},{'H','P'},{'Q','K'}

It is compared values in each sub-list and swap them (if necessary) in the original array.
'S' > 'C' ? true – swap them D = {'C','B','I','M','H','Q','S','L','R','E','P','K'}
'B' > 'L' ? false
'I' > 'R' ? false
'M' > 'E' ? true – swap them D = {'C','B','I','E','H','Q','S','L','R','M','P','K'}
'H' > 'P' ? false

'Q' > 'K' ? true – swap them D = {'C','B','I','E','H','K','S','L','R','M','P','Q'}

Then calculate gap again.
pass = 2
gap = floor(gap/2) = floor(6/2) = floor(3) = 3

D = {'C','B','I','E','H','K','S','L','R','M','P','Q'}
It is made a **virtual** sub-list of all values located at the interval of gap positions.
{'C','E','S','M'},{'B','H','L','P'},{'I','K','R','Q'}

It is compared values in each sub-list and swap them (if necessary) in the original array.
'C' > 'E' ? false
'B' > 'H' ? false
'I' > 'K' ? false
'E' > 'S' ? false
'H' > 'L' ? false
'K' > 'R' ? false
'S' > 'M' ? true – swap them D = {'C','B','I','E','H','K','M','L','R','S','P','Q'}
'E' > 'M' ? false
'L' > 'P' ? false
'R' > 'Q' ? true – swap them D = {'C','B','I','E','H','K','M','L','Q','S','P','R'}
'K' > 'Q' ? false

Then calculate gap again.
pass = 3
gap = floor(gap/2) = floor(3/2) = floor(1,5) = 1

D = {'C','B','I','E','H','K','M','L','Q','S','P','R'}
Finally, we sort the rest of the array using interval of value 1. Shell sort uses insertion sort to sort the array.

'C' > 'B' ? true – swap them D = {'B','C','I','E','H','K','M','L','Q','S','P','R'}
'C' > 'I' ? false
'I' > 'E' ? true – swap them D = {'B','C','E','I','H','K','M','L','Q','S','P','R'}
'C' > 'E' ? false
'I' > 'H' ? true – swap them D = {'B','C','E','H','I','K','M','L','Q','S','P','R'}
'E' > 'H' ? false
'I' > 'K' ? false
'K' > 'M' ? false
'M' > 'L' ? true – swap them D = {'B','C','E','H','I','K','L','M','Q','S','P','R'}
'K' > 'L' ? false
'M' > 'Q' ? false
'Q' > 'S' ? false
'S' > 'P' ? true – swap them D = {'B','C','E','H','I','K','L','M','Q','P','S','R'}
'Q' > 'P' ? true – swap them D = {'B','C','E','H','I','K','L','M','P','Q','S','R'}
'M' > 'P' ? false
'S' > 'R' ? true – swap them D = {'B','C','E','H','I','K','L','M','P','Q','R','S'}
'Q' > 'R' ? false

Then calculate gap again
gap = floor(gap/2) = floor(1/2) = floor(0,5) = 0
And shell sort algroitm is over. Array was sorted.

Size : 12
Total comparision : 34
Total swap : 12

Merge Sort:

<table>
<tr><td colspan="12">‘S’,’B’,’I’,’M’,’H’,’Q’,’C’,’L’,’R’,’E’,’P’,’K’</td></tr>
<tr><td colspan="12">Size = 12
l(left) = 0
r(right) = size -1 = 11
l < r, divide array by two
m(middle) = floor((l+r)/2) = 5</td></tr>
<tr><td colspan="6">‘S’,’B’,’I’,’M’,’H’,’Q’</td><td colspan="6">’C’,’L’,’R’,’E’,’P’,’K’</td></tr>
<tr><td colspan="6">l = 0
r = 5
l < r
m = 2</td><td colspan="6">l = 6
r = 11
l < r
m = 8</td></tr>
<tr><td colspan="3">‘S’,’B’,’I’</td><td colspan="3">’M’,’H’,’Q’</td><td colspan="3">’C’,’L’,’R’</td><td colspan="3">’E’,’P’,’K’</td></tr>
<tr><td colspan="3">l = 0
r = 2
l < r
m = 1</td><td colspan="3">l = 3
r = 5
l < r
m = 4</td><td colspan="3">l = 6
r = 8
l < r
m = 7</td><td colspan="3">l = 9
r = 11
l < r
m = 10</td></tr>
<tr><td colspan="2">‘S’,’B’</td><td>’I’</td><td colspan="2">’M’,’H’</td><td>’Q’</td><td colspan="2">’C’,’L’</td><td>’R’</td><td colspan="2">’E’,’P’</td><td>’K’</td></tr>
<tr><td colspan="2">l = 0
r = 1
l < r
m = 0</td><td>l = 2
r = 2
l >=r</td><td colspan="2">l = 3
r = 4
l < r
m = 3</td><td>l = 5
r = 5
l >=r</td><td colspan="2">l = 6
r = 7
l < r
m = 6</td><td>l = 8
r = 8
l >=r</td><td colspan="2">l = 9
r = 10
l < r
m = 9</td><td>l =11
r =11
l >=r</td></tr>
<tr><td>‘S’</td><td>‘B’</td><td>‘I’</td><td>‘M’</td><td>’H’</td><td>’Q’</td><td>’C’</td><td>’L’</td><td>’R’</td><td>’E’</td><td>’P’</td><td>’K’</td></tr>
<tr><td>l = 0
r = 0
l >=r</td><td>l = 1
r = 1
l >=r</td><td></td><td>l = 3
r = 3
l >=r</td><td>l = 4
r = 4
l >=r</td><td></td><td>l = 6
r = 6
l >=r</td><td>l = 7
r = 7
l >=r</td><td></td><td>l = 9
r = 9
l >=r</td><td>l =10
r =10
l >=r</td><td></td></tr>
<tr><td>‘S’</td><td>‘B’</td><td>‘I’</td><td>‘M’</td><td>’H’</td><td>’Q’</td><td>’C’</td><td>’L’</td><td>’R’</td><td>’E’</td><td>’P’</td><td>’K’</td></tr>
<tr><td colspan="2">‘S’ < ‘B’ ?
false</td><td></td><td colspan="2">‘M’ < ‘H’ ?
false</td><td></td><td colspan="2">‘C’ < ‘L’ ?
true</td><td></td><td colspan="2">‘E’ < ‘P’ ?
true</td><td></td></tr>
<tr><td colspan="2">‘B’,’S’</td><td>‘I’</td><td colspan="2">‘H’,’M’</td><td>‘Q’</td><td colspan="2">‘C’,’L’</td><td>‘R’</td><td colspan="2">‘E’,’P’</td><td>‘K’</td></tr>
<tr><td colspan="3">‘B’ < ‘I’ ? true
‘S’ < ‘I’ ? false</td><td colspan="3">‘H’ < ‘Q’ ? true
‘M’ < ‘Q’ ? true</td><td colspan="3">‘C’ < ‘R’ ? true
‘L’ < ‘R’ ? true</td><td colspan="3">‘E’ < ‘K’ ? true
‘P’ < ‘K’ ? false</td></tr>
<tr><td colspan="3">‘B’,’I’,’S’</td><td colspan="3">‘H’,’M’,’Q’</td><td colspan="3">‘C’,’L’,’R’</td><td colspan="3">‘E’,’K’,’P’</td></tr>
<tr><td colspan="6">‘B’ < ‘H’ ? true
‘I’ < ‘H’ ? false
‘I’ < ‘M’ ? true
‘S’ < ‘M’ ? false
‘S’ < ‘Q’ ? false</td><td colspan="6">‘C’ < ‘E’ ? true
‘L’ < ‘E’ ? false
‘L’ < ‘K’ ? false
‘L’ < ‘P’ ? true
‘R’ < ‘P’ ? false</td></tr>
<tr><td colspan="6">‘B’,’H’,’I’,’M’,’Q’,’S’</td><td colspan="6">‘C’,’E’,’K’,’L’,’P’,’R’</td></tr>
<tr><td colspan="12">‘B’ < ‘C’ ? true</td></tr>
</table>

‘H’ < ‘C’ ? false ‘H’ < ‘E’ ? false ‘H’ < ‘K’ ? true ‘I’ < ‘K’ ? true ‘M’ < ‘K’ ? false ‘M’ < ‘L’ ? false ‘M’ < ‘P’ ? true ‘Q’ < ‘P’ ? false ‘Q’ < ‘R’ ? true ‘S’ < ‘R’ ? false
‘B’,’C’,’E’,’H’,’I’,’K’,’L’,’M’,’P’,’Q’,’R’,’S’

Total comparision : 56 (base case + comprision between array elements)
Total displacement : 44 (replace elements a new array)

Heap Sort:

Note : The quotes aren’t written to fit the table.

D = {‘S’,’B’,’I’,’M’,’H’,’Q’,’C’,’L’,’R’,’E’,’P’,’K’}

Build a Max Heap array

Algorithm: In the default array, the value corresponding to the array index is compared to the value corresponding to the parent index ((array index-1) / 2) in the max heap array. If the operation is true, their positions are swapped. This process continues until the comparision operation is false or has no parent. The general process continues until array index is smaller than size.

Array Index	Max Heap Array	Parent Index	Description	After operation Max Heap Array
0	S,B,I,M,H,Q,C,L,R,E,P,K	-	Initially 0. index is root	S,B,I,M,H,Q,C,L,R,E,P,K
1	S,B,I,M,H,Q,C,L,R,E,P,K	0	B > S ? false	S,B,I,M,H,Q,C,L,R,E,P,K
2	S,B,I,M,H,Q,C,L,R,E,P,K	0	I > S ? false	S,B,I,M,H,Q,C,L,R,E,P,K
3	S,B,I,M,H,Q,C,L,R,E,P,K	1	M > B ? true – swap them	S,M,I,B,H,Q,C,L,R,E,P,K
3	S,M,I,B,H,Q,C,L,R,E,P,K	0	M > S ? false	S,M,I,B,H,Q,C,L,R,E,P,K
4	S,M,I,B,H,Q,C,L,R,E,P,K	1	H > M ? false	S,M,I,B,H,Q,C,L,R,E,P,K
5	S,M,I,B,H,Q,C,L,R,E,P,K	2	Q > I ? true – swap them	S,M,Q,B,H,I,C,L,R,E,P,K
5	S,M,Q,B,H,I,C,L,R,E,P,K	0	Q > S ? false	S,M,Q,B,H,I,C,L,R,E,P,K
6	S,M,Q,B,H,I,C,L,R,E,P,K	2	C > Q ? false	S,M,Q,B,H,I,C,L,R,E,P,K
7	S,M,Q,B,H,I,C,L,R,E,P,K	3	L > B ? true – swap them	S,M,Q,L,H,I,C,B,R,E,P,K
7	S,M,Q,L,H,I,C,B,R,E,P,K	1	L > M ? false	S,M,Q,L,H,I,C,B,R,E,P,K
8	S,M,Q,L,H,I,C,B,R,E,P,K	3	R > L ? true – swap them	S,M,Q,R,H,I,C,B,L,E,P,K
8	S,M,Q,R,H,I,C,B,L,E,P,K	1	R > M ? true – swap them	S,R,Q,M,H,I,C,B,L,E,P,K
8	S,R,Q,M,H,I,C,B,L,E,P,K	0	R > S ? false	S,R,Q,M,H,I,C,B,L,E,P,K
9	S,R,Q,M,H,I,C,B,L,E,P,K	4	E > H ? false	S,R,Q,M,H,I,C,B,L,E,P,K
10	S,R,Q,M,H,I,C,B,L,E,P,K	4	P > H ? true – swap them	S,R,Q,M,P,I,C,B,L,E,H,K
10	S,R,Q,M,P,I,C,B,L,E,H,K	1	P > R ? false	S,R,Q,M,P,I,C,B,L,E,H,K
11	S,R,Q,M,P,I,C,B,L,E,H,K	5	K > I ? true – swap them	S,R,Q,M,P,K,C,B,L,E,H,I
11	S,R,Q,M,P,K,C,B,L,E,H,I	2	K > Q ? false	S,R,Q,M,P,K,C,B,L,E,H,I
12			Max heap array is created	S,R,Q,M,P,K,C,B,L,E,H,I

D = {'S','R','Q','M','P','K','C','B','L','E','H','I' }			
Sorting			
Algorithm: Swap array index with head in array. Heapify is applied to the part up to the array index in the array. Sorting is done until the array index is 0.			
Array Index	**Array**	**Description**	**After operation array**
11	S,R,Q,M,P,K,C,B,L,E,H,I	Head and array index were swapped. Apply heapify	I,R,Q,M,P,K,C,B,L,E,H,S
	I,R,Q,M,P,K,C,B,L,E,H,S	R > Q ? true max child = R I < R ? true – swap them	R,I,Q,M,P,K,C,B,L,E,H,S
	R,I,Q,M,P,K,C,B,L,E,H,S	M > P ? false max child = P I < P ? true – swap them	R,P,Q,M,I,K,C,B,L,E,H,S
	R,P,Q,M,I,K,C,B,L,E,H,S	E > H ? false max child = H I < H ? false	R,P,Q,M,I,K,C,B,L,E,H,S
10	R,P,Q,M,I,K,C,B,L,E,H,S	Head and array index were swapped. Apply heapify	H,P,Q,M,I,K,C,B,L,E,R,S
	H,P,Q,M,I,K,C,B,L,E,R,S	P > Q ? false max child = Q H < Q ? true – swap them	Q,P,H,M,I,K,C,B,L,E,R,S
	Q,P,H,M,I,K,C,B,L,E,R,S	K > C ? true max child = K H < K ? true – swap them	Q,P,K,M,I,H,C,B,L,E,R,S
	Q,P,K,M,I,H,C,B,L,E,R,S	Element H doesn't have children	Q,P,K,M,I,H,C,B,L,E,R,S
9	Q,P,K,M,I,H,C,B,L,E,R,S	Head and array index were swapped. Apply heapify	E,P,K,M,I,H,C,B,L,Q,R,S
	E,P,K,M,I,H,C,B,L,Q,R,S	P > K ? true max child = P E < P ? true – swap them	P,E,K,M,I,H,C,B,L,Q,R,S
	P,E,K,M,I,H,C,B,L,Q,R,S	M > I ? true max child = M E < M ? true – swap them	P,M,K,E,I,H,C,B,L,Q,R,S
	P,M,K,E,I,H,C,B,L,Q,R,S	B > L ? false max child = L E < L ? true – swap them	P,M,K,L,I,H,C,B,E,Q,R,S
	P,M,K,L,I,H,C,B,E,Q,R,S	Element E doesn't have children	P,M,K,L,I,H,C,B,E,Q,R,S
8	P,M,K,L,I,H,C,B,E,Q,R,S	Head and array index were swapped. Apply heapify	E,M,K,L,I,H,C,B,P,Q,R,S
	E,M,K,L,I,H,C,B,P,Q,R,S	M > K ? true max child = M E < M ? true – swap them	M,E,K,L,I,H,C,B,P,Q,R,S
	M,E,K,L,I,H,C,B,P,Q,R,S	L > I ? true max child = L E < L ? true – swap them	M,L,K,E,I,H,C,B,P,Q,R,S
	M,L,K,E,I,H,C,B,P,Q,R,S	Max child = B E < B ? false – swap them	M,L,K,B,I,H,C,E,P,Q,R,S
	M,L,K,B,I,H,C,E,P,Q,R,S	Element E doesn't have children	M,L,K,B,I,H,C,E,P,Q,R,S
7	M,L,K,B,I,H,C,E,P,Q,R,S	Head and array index were swapped. Apply heapify	E,L,K,B,I,H,C,M,P,Q,R,S
	E,L,K,B,I,H,C,M,P,Q,R,S	L > K ? true max child = L E < L ? true – swap them	L,E,K,B,I,H,C,M,P,Q,R,S
	L,E,K,B,I,H,C,M,P,Q,R,S	B > I ? false max child = I E < I ? true – swap them	L,I,K,B,E,H,C,M,P,Q,R,S
	L,I,K,B,E,H,C,M,P,Q,R,S	Element E doesn't have children	L,I,K,B,E,H,C,M,P,Q,R,S
6	L,I,K,B,E,H,C,M,P,Q,R,S	Head and array index were swapped. Apply heapify	C,I,K,B,E,H,L,M,P,Q,R,S
	C,I,K,B,E,H,L,M,P,Q,R,S	I > K ? false max child = K C < K ? true – swap them	K,I,C,B,E,H,L,M,P,Q,R,S

	K,I,C,B,E,H,L,M,P,Q,R,S	Max child = H C < H ? true – swap them	K,I,H,B,E,C,L,M,P,Q,R,S
	K,I,H,B,E,C,L,M,P,Q,R,S	Element C doesn't have children	K,I,H,B,E,C,L,M,P,Q,R,S
5	K,I,H,B,E,C,L,M,P,Q,R,S	Head and array index were swapped. Apply heapify	C,I,H,B,E,K,L,M,P,Q,R,S
	C,I,H,B,E,K,L,M,P,Q,R,S	I > H ? true max child = I C < I ? true – swap them	I,C,H,B,E,K,L,M,P,Q,R,S
	I,C,H,B,E,K,L,M,P,Q,R,S	B > E ? false max child = E C < E ? true – swap them	I,E,H,B,C,K,L,M,P,Q,R,S
	I,E,H,B,C,K,L,M,P,Q,R,S	Element C doesn't have children	I,E,H,B,C,K,L,M,P,Q,R,S
4	I,E,H,B,C,K,L,M,P,Q,R,S	Head and array index were swapped. Apply heapify	C,E,H,B,I,K,L,M,P,Q,R,S
	C,E,H,B,I,K,L,M,P,Q,R,S	E > H ? false max child = H C < H ? true – swap them	H,E,C,B,I,K,L,M,P,Q,R,S
	H,E,C,B,I,K,L,M,P,Q,R,S	Element C doesn't have children	H,E,C,B,I,K,L,M,P,Q,R,S
3	H,E,C,B,I,K,L,M,P,Q,R,S	Head and array index were swapped.	B,E,C,H,I,K,L,M,P,Q,R,S
	B,E,C,H,I,K,L,M,P,Q,R,S	E > C ? true max child = E B < E ? true – swap them	E,B,C,H,I,K,L,M,P,Q,R,S
	E,B,C,H,I,K,L,M,P,Q,R,S	Element B doesn't have children	E,B,C,H,I,K,L,M,P,Q,R,S
2	E,B,C,H,I,K,L,M,P,Q,R,S	Head and array index were swapped.	C,B,E,H,I,K,L,M,P,Q,R,S
	C,B,E,H,I,K,L,M,P,Q,R,S	Max child = B C < B ? false	C,B,E,H,I,K,L,M,P,Q,R,S
1	C,B,E,H,I,K,L,M,P,Q,R,S	Head and array index were swapped.	B,C,E,H,I,K,L,M,P,Q,R,S
	B,C,E,H,I,K,L,M,P,Q,R,S	Element B doesn't have children	B,C,E,H,I,K,L,M,P,Q,R,S
0		Array is sorted	B,C,E,H,I,K,L,M,P,Q,R,S

Total comparision : 67 (build + sort)
Total displacement : 36 (swap amount)

Quick Sort:

Pivot: Always picked last element as pivot
Partition Algorithm
The logic is simple, we start from the leftmost element and keep track of index of smaller elements as i. While traversing, if we find a smaller element, we swap current element with arr[i]. Otherwise we ignore current element.
Note: The quotes aren't written to fit the table.

D = {'S','B','I','M','H','Q','C','L','R','E','P','K'}

Array	Pivot Value	p	i	end	Description	After operation array
S,B,I,M,H,Q,C,L,R,E,P,K			0	11	i < end ? true – partition calls.	
S,B,I,M,H,Q,C,L,R,E,P,K	K	-1	0	11	S < K ? false	
S,B,I,M,H,Q,C,L,R,E,P,K	K	-1	1	11	B < K ? true, ++p , swap p and i (S,B)	B,S,I,M,H,Q,C,L,R,E,P,K
B,S,I,M,H,Q,C,L,R,E,P,K	K	0	2	11	I < K ? true, ++p, swap p and i (S,I)	B,I,S,M,H,Q,C,L,R,E,P,K
B,I,S,M,H,Q,C,L,R,E,P,K	K	1	3	11	M < K ? false	
B,I,S,M,H,Q,C,L,R,E,P,K	K	1	4	11	H < K ? true, ++p, swap p and i (S,H)	B,I,H,M,S,Q,C,L,R,E,P,K
B,I,H,M,S,Q,C,L,R,E,P,K	K	2	5	11	Q < K ? false	
B,I,H,M,S,Q,C,L,R,E,P,K	K	2	6	11	C < K ? true, ++p, swap p and i (M,C)	B,I,H,C,S,Q,M,L,R,E,P,K

B,I,H,C,S,Q,M,L,R,E,P,K	K	3	7	11	L < K ? false	
B,I,H,C,S,Q,M,L,R,E,P,K	K	3	8	11	R < K ? false	
B,I,H,C,S,Q,M,L,R,E,P,K	K	3	9	11	E < K ? true, ++p, swap p and i (S,E)	B,I,H,C,E,Q,M,L,R,S,P,K
B,I,H,C,E,Q,M,L,R,S,P,K	K	4	10	11	P < K ? false	
B,I,H,C,E,Q,M,L,R,S,P,K	K	4	11	11	i < end ? false. Last operation ++p and swap p and pivot (Q,K). Partition is finish. So next recursion calls Left part: i = 0, end(p-1) = 4 Right part: i(p+1) = 6, end = 11	B,I,H,C,E,K,M,L,R,S,P,Q
B,I,H,C,E,K,M,L,R,S,P,Q			0	4	i < end ? true – partition calls.	
B,I,H,C,E,K,M,L,R,S,P,Q	E	-1	0	4	B < E ? true, ++p, swap p and i (B,B)	B,I,H,C,E,K,M,L,R,S,P,Q
B,I,H,C,E,K,M,L,R,S,P,Q	E	0	1	4	I < E ? false	
B,I,H,C,E,K,M,L,R,S,P,Q	E	0	2	4	H < E ? false	
B,I,H,C,E,K,M,L,R,S,P,Q	E	0	3	4	C < E ? true, ++p, swap p and i (I,C)	B,C,H,I,E,K,M,L,R,S,P,Q
B,C,H,I,E,K,M,L,R,S,P,Q	E	1	4	4	i < end ? false. Last operation ++p and swap p and pivot (H,E). Partition is finish. So next recursion calls Left part: i = 0, end(p-1) = 1 Right part: i(p+1) = 3, end = 4	B,C,E,I,H,K,M,L,R,S,P,Q
B,C,E,I,H,K,M,L,R,S,P,Q			0	1	i < end ? true – partition calls.	
B,C,E,I,H,K,M,L,R,S,P,Q	C	-1	0	1	B < C ? true, ++p, swap p and i (B,B)	B,C,E,I,H,K,M,L,R,S,P,Q
B,C,E,I,H,K,M,L,R,S,P,Q	C	0	1	1	i < end ? false. Last operation ++p and swap p and pivot (C,C). Partition is finish. So next recursion calls Left part: i = 0, end(p-1) = 0 Right part: i(p+1) = 2, end = 1	B,C,E,I,H,K,M,L,R,S,P,Q
B,C,E,I,H,K,M,L,R,S,P,Q			0	0	i < end ? false	
B,C,E,I,H,K,M,L,R,S,P,Q			2	1	i < end ? false	
B,C,E,I,H,K,M,L,R,S,P,Q			3	4	i < end ? true – partition calls.	
B,C,E,I,H,K,M,L,R,S,P,Q	H	2	3	4	I < H ? false	
B,C,E,I,H,K,M,L,R,S,P,Q	H	2	4	4	i < end ? false. Last operation ++p and swap p and pivot (I,H). Partition is finish. So next recursion calls Left part: i = 3, end(p-1) = 2 Right part: i(p+1) = 4, end = 4	B,C,E,H,I,K,M,L,R,S,P,Q
B,C,E,H,I,K,M,L,R,S,P,Q			3	2	i < end ? false	
B,C,E,H,I,K,M,L,R,S,P,Q			4	4	i < end ? false	
B,C,E,H,I,K,M,L,R,S,P,Q			6	11	i < end ? true – partition calls.	
B,C,E,H,I,K,M,L,R,S,P,Q	Q	5	6	11	M < Q ? true, ++p, swap p and i (M,M)	B,C,E,H,I,K,M,L,R,S,P,Q
B,C,E,H,I,K,M,L,R,S,P,Q	Q	6	7	11	L < Q ? false, ++p, swap p and i (L,L)	
B,C,E,H,I,K,M,L,R,S,P,Q	Q	7	8	11	R < Q ? false	
B,C,E,H,I,K,M,L,R,S,P,Q	Q	7	9	11	S < Q ? false	
B,C,E,H,I,K,M,L,R,S,P,Q	Q	7	10	11	P < Q ? true, ++p, swap p and i (R,P)	B,C,E,H,I,K,M,L,P,S,R,Q
B,C,E,H,I,K,M,L,P,S,R,Q	Q	8	11	11	i < end ? false. Last operation ++p and swap p and pivot (S,Q). Partition is finish. So next recursion calls Left part: i = 6, end(p-1) = 8 Right part: i(p+1) = 10, end = 11	B,C,E,H,I,K,M,L,P,Q,R,S
B,C,E,H,I,K,M,L,P,Q,R,S			6	8	i < end ? true – partition calls.	
B,C,E,H,I,K,M,L,P,Q,R,S	P	5	6	8	M < P ? true, ++p, swap p and i (M,M)	B,C,E,H,I,K,M,L,P,Q,R,S

B,C,E,H,I,K,M,L,P,Q,R,S	P	6	7	8	L < P ? true, ++p, swap p and i (L,L)	B,C,E,H,I,K,M,L,P,Q,R,S
B,C,E,H,I,K,M,L,P,Q,R,S	P	7	8	8	i < end ? false. Last operation ++p and swap p and pivot (P,P). Partition is finish. So next recursion calls Left part: i = 6, end(p-1) = 7 Right part: i(p+1) = 9, end = 8	B,C,E,H,I,K,M,L,P,Q,R,S
B,C,E,H,I,K,M,L,P,Q,R,S			6	7	i < end ? true – partition calls.	
B,C,E,H,I,K,M,L,P,Q,R,S	L	5	6	7	M < L ? false	
B,C,E,H,I,K,M,L,P,Q,R,S	L	5	7	7	i < end ? false. Last operation ++p and swap p and pivot (M,L). Partition is finish. So next recursion calls Left part: i = 6, end(p-1) = 5 Right part: i(p+1) = 7, end = 8	B,C,E,H,I,K,L,M,P,Q,R,S
B,C,E,H,I,K,L,M,P,Q,R,S			6	5	i < end ? false	
B,C,E,H,I,K,L,M,P,Q,R,S			7	8	i < end ? true – partition calls.	
B,C,E,H,I,K,L,M,P,Q,R,S	P	6	7	8	M < P ? true, ++p, swap p and i (M,M)	B,C,E,H,I,K,L,M,P,Q,R,S
B,C,E,H,I,K,L,M,P,Q,R,S	P	7	8	8	i < end ? false. Last operation ++p and swap p and pivot (L,L). Partition is finish. So next recursion calls Left part: i = 7, end(p-1) = 7 Right part: i(p+1) = 9, end = 8	B,C,E,H,I,K,L,M,P,Q,R,S
B,C,E,H,I,K,L,M,P,Q,R,S			7	7	i < end ? false	
B,C,E,H,I,K,L,M,P,Q,R,S			9	8	i < end ? false	
B,C,E,H,I,K,L,M,P,Q,R,S			9	8	i < end ? false	
B,C,E,H,I,K,L,M,P,Q,R,S			10	11	i < end ? true – partition calls.	
B,C,E,H,I,K,L,M,P,Q,R,S	S	9	10	11	R < S ? true, ++p, swap p and i (R,R)	B,C,E,H,I,K,L,M,P,Q,R,S
B,C,E,H,I,K,L,M,P,Q,R,S	S	10	11	11	i < end ? false. Last operation ++p and swap p and pivot (S,S). Partition is finish. So next recursion calls Left part: i = 10, end(p-1) = 10 Right part: i(p+1) = 12, end = 11	B,C,E,H,I,K,L,M,P,Q,R,S
B,C,E,H,I,K,L,M,P,Q,R,S			10	10	i < end ? false	
B,C,E,H,I,K,L,M,P,Q,R,S			12	11	i < end ? false	
					Array is sorted	B,C,E,H,I,K,L,M,P,Q,R,S

Total comparision : 55 (base case + comparision between array elements)
Total displacement : 24 (swap amount)