Alexandria University
Faculty of Engineering

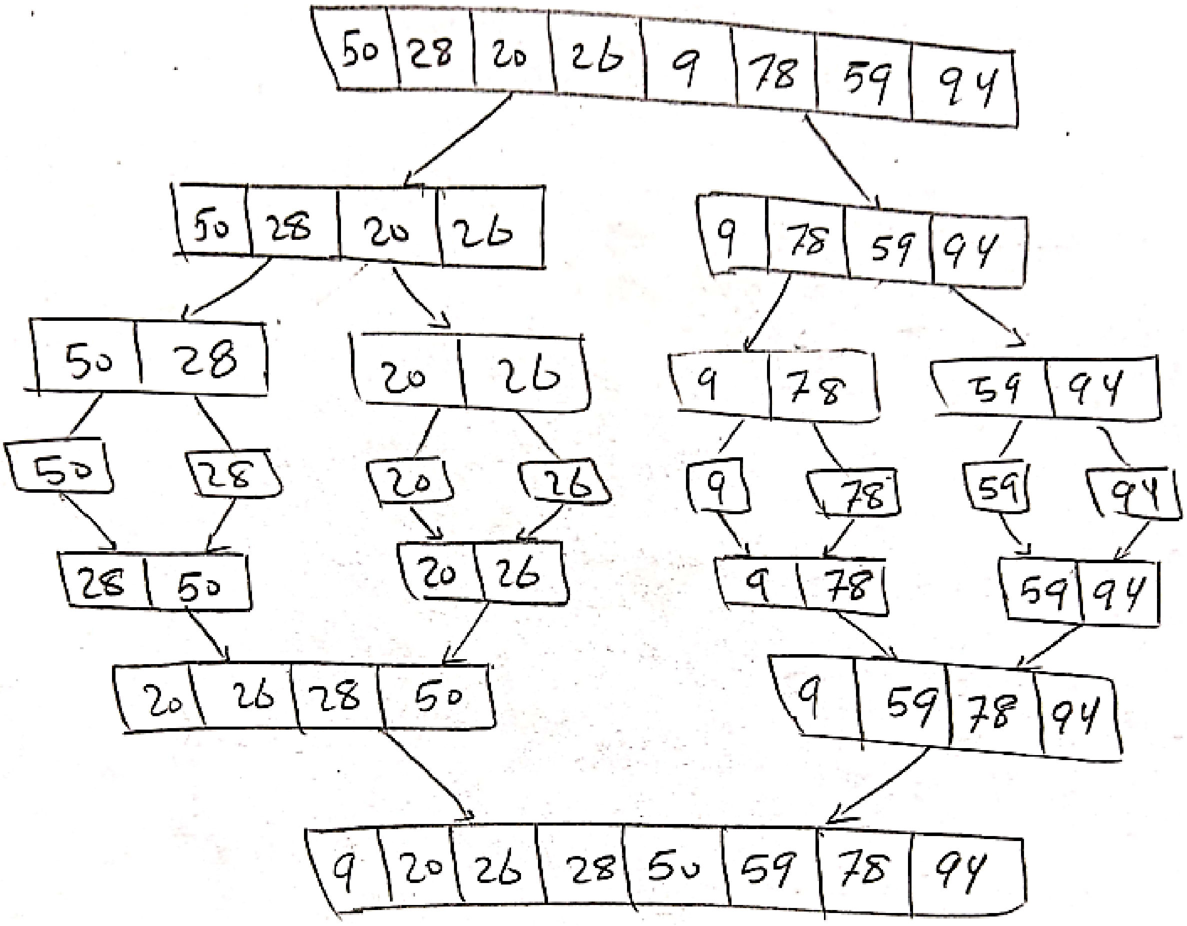
Assignment NO. 4

By:

Ali Mohamed Taima Hummus

Question(1):

a)



---

b)

0	1	2	3	4	5	6	7
50	28	20	26	9	78	59	94

- I will start to find(first, middle, last):
  - First = arr[0] = 50
  - Middle = arr[3] = 26
  - Last = arr[7] = 94
- Sorting first, middle, last
  - First < last →sorted
  - First > middle →swap(50, 26)
  - Middle < last → sorted

0	1	2	3	4	5	6	7
26	28	20	50	9	78	59	94

- First I will the array approximately (allocate the large elements in the right subarray and the small values in the left subarray)
  - I will choose the pivot as the median
    - Pivot = arr[3] = 50
  - I will swap the pivot with the before last element, as the last elements is already sorted with the middle in the last step

- I will assume variable a that initialize with index 0 and variable b that initialize with the index of the pivot

a = 0	1	2	3	4	5	b = 6	7
26	28	20	59	9	78	Pivot = 50	94

- I will compare array[a] with array[b], if arr[a] < arr[b] then I will increment a, and if arr[a] > arr[b] then I will decrement b and compare again ….. and so on
  - arr[0] < arr[6] →sorted →a = 1
  - arr[1] < arr[6] →sorted →a = 2
  - arr[2] < arr[6] →sorted →a = 3
  - arr[3] > arr[6] →unsorted →b = 5
  - arr[3] > arr[5] →unsorted →b = 4
  - arr[3] > arr[4] →unsorted →if we decrement b again, it will be equal to a, so we will swap the values of a & b
  - arr[a] > pivot, so now swap element in a with the pivot

0	1	2	3	4	5	6	7
26	28	20	9	50	78	59	94

- Now we locate the small elements in left subarray and the large elements in right subarray, and the next step is to sort right and left subarrays
- For left sub array from index(0) to index(3)
  - I will apply the same concept in the main array
  - I will start to find(first, middle, last):
    - First = 26
    - Middle = 28
    - Last = 9

0	1	2	3
26	28	20	9

  - Sorting first, middle, last
    - First > last →swap(9, 26)
      - First = 9, last = 26

0	1	2	3
9	28	20	26

    - First < middle →sorted
    - Middle > last →swap(28, 26)

0	1	2	3
9	26	20	28

  - The pivot is the median, pivot = 26
  - Swap pivot with the element of before last (20)
  - Assume a = 0, b = 2 (index of pivot)

a = 0	1	b = 2	3
9	20	Pivot = 26	28

  - I will compare array[a] with array[b], if arr[a] < arr[b] then I will increment a, and if arr[a] > arr[b] then I will decrement b and compare again ….. and so on
    - arr[0] < arr[2] →sorted →a = 1
    - arr[1] < arr[2] →sorted → if we increment a again, it will be equal to b, so we will stop
    - arr[a] < pivot →sorted

- for left subarray is sorted

0	1
9	20

- For right subarray is sorted

2	3
26	28

- So all the left subarray is sorted now

0	1	2	3
9	20	26	28

- For right subarray from index(4) to index(7)
    - I will apply the same concept in the main array
    - I will start to find(first, middle, last):
        - First = 50
        - Middle = 78
        - Last = 94

4	5	6	7
50	78	59	94

    - Sorting first, middle, last
        - First < last →sorted
        - First < middle →sorted
        - Middle < last →sorted
    - The pivot is the median, pivot = 78
    - Swap pivot with the element of before last (59)
    - Assume a = 4, b = 6 (index of pivot)

4	5	6	7
50	59	Pivot = 78	94

    -
    - I will compare array[a] with array[b], if arr[a] < arr[b] then I will increment a, and if arr[a] > arr[b] then I will decrement b and compare again ….. and so on
        - arr[4] < arr[6] →sorted →a = 5
        - arr[5] < arr[6] →sorted → if we increment a again, it will be equal to b, so we will stop arr[a] < pivot →sorted
    - for left subarray is sorted

4	5
9	59

    - For right subarray is sorted

6	7
78	94

    - So all the right subarray is sorted now

4	5	6	7
9	59	78	94

❖ **Whole array is now sorted**

c)

Because quick sort is in place algorithm while merge sort is external algorithm which means that quick sort doesn't require any extra memory while merge sort require memory (require according to the data size)

Question(2):

a)

0	1	2	3	4	5	6	7	8	9	10	11	12
6	56	34	23	16	4	8	1	30	41	37	52	2

① gap = 13/2 = 6

1- start with 0th arr[(6-gap)=0] & 6th arr[gap=6]

2- 1st arr[(7-gap)=1] & 7th arr[gap+1=7]

3- 2nd arr[(8-gap)=2] & 8th arr[gap+2=8]

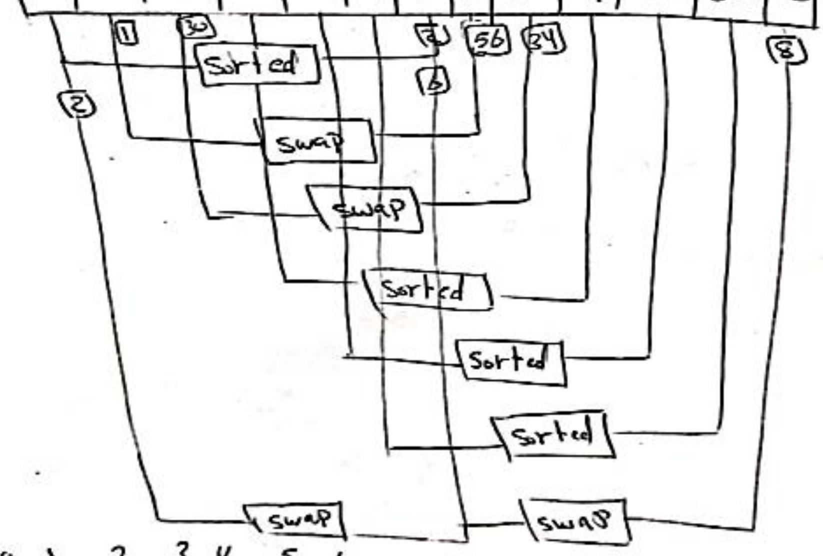
4- 3rd arr[(9-gap)=3] & 9th arr[gap+3=9]

5- 4th arr[(10-gap)=4] & 10th arr[gap+4=10]

6- 5th arr[(11-gap)=5] & 11th arr[gap+5=11]

7- 6th arr[(12-gap)=6] & 12th arr[gap+6=12]

0	1	2	3	4	5	6	7	8	9	10	11	12
6	56	34	23	16	4	8	1	30	41	37	52	2



0	1	2	3	4	5	6	7	8	9	10	11	12
2	1	30	23	16	4	6	56	34	41	37	52	8

② gap = 13/4 = 3

1- 0th arr[3-3=0] & 3rd arr[gap=3]

2- arr[4-3=1] & arr[gap+1=4]

3- arr[5-3=2] & arr[gap+2=5]

4- arr[6-3=3] & arr[gap+3=6]

5- arr[7-3=4] & arr[gap+4=7]

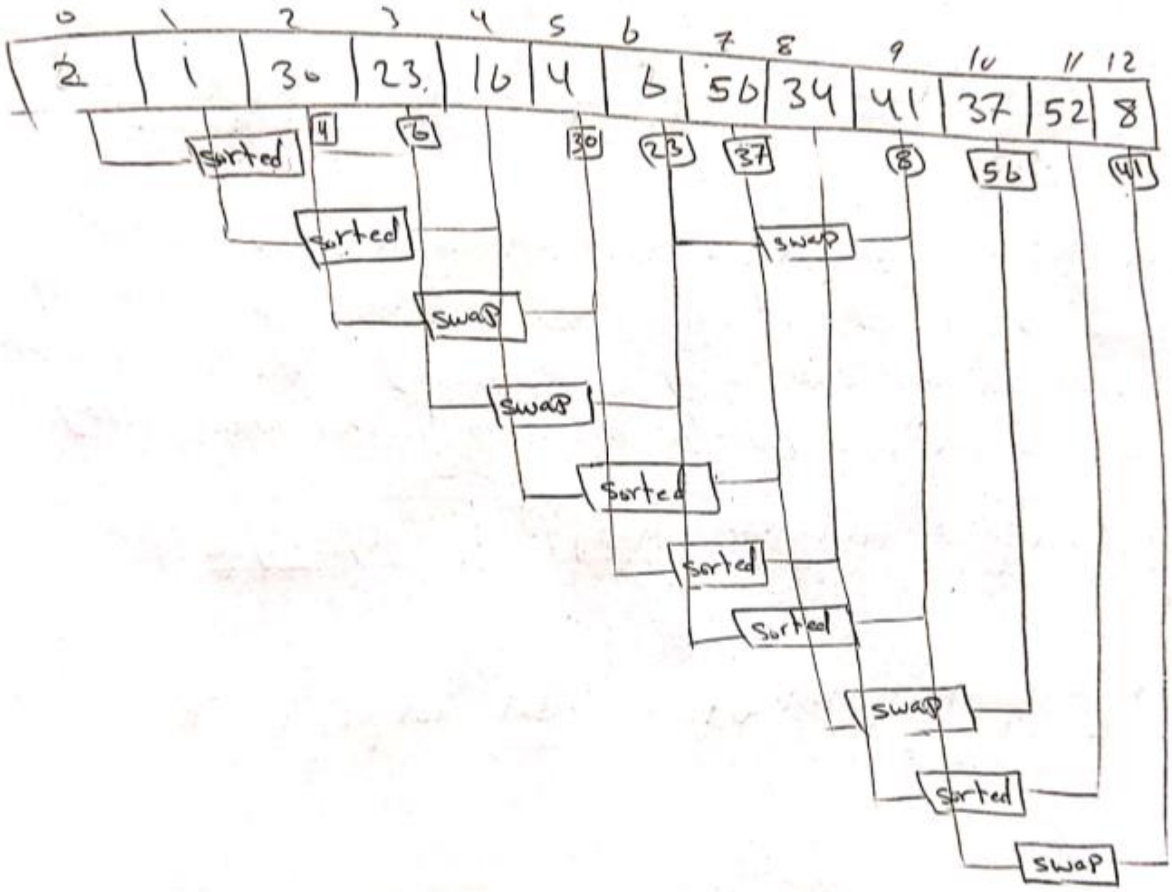
6. arr[8-3 = 5] & arr[gap+5 = 8]

7- arr[9-3 = 6] & arr[gap+6 = 9]

8- arr[10-3 = 7] & arr[gap+7 = 10]

9- arr[11-3 = 8] & arr[gap+8 = 11]

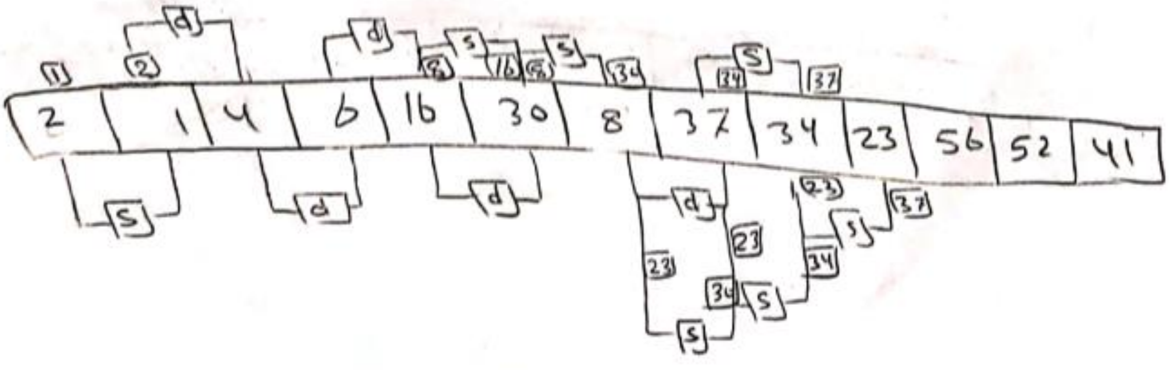
10- arr[12-3 = 9] & arr[gap+9 = 12]



2	1	4	6	16	30	8	37	34	23	56	52	41

(3) gap = 13/8 = 1

swap each element with its next:- swap(S) sorted(d)



- For 8 I make 2 swaPes to Put it in the right Position (8 & 30), (8 & 16)
- For 23 I make 3 swaPes to Put it in the right Position (23 & 37), (23 & 34), (23 & 30)

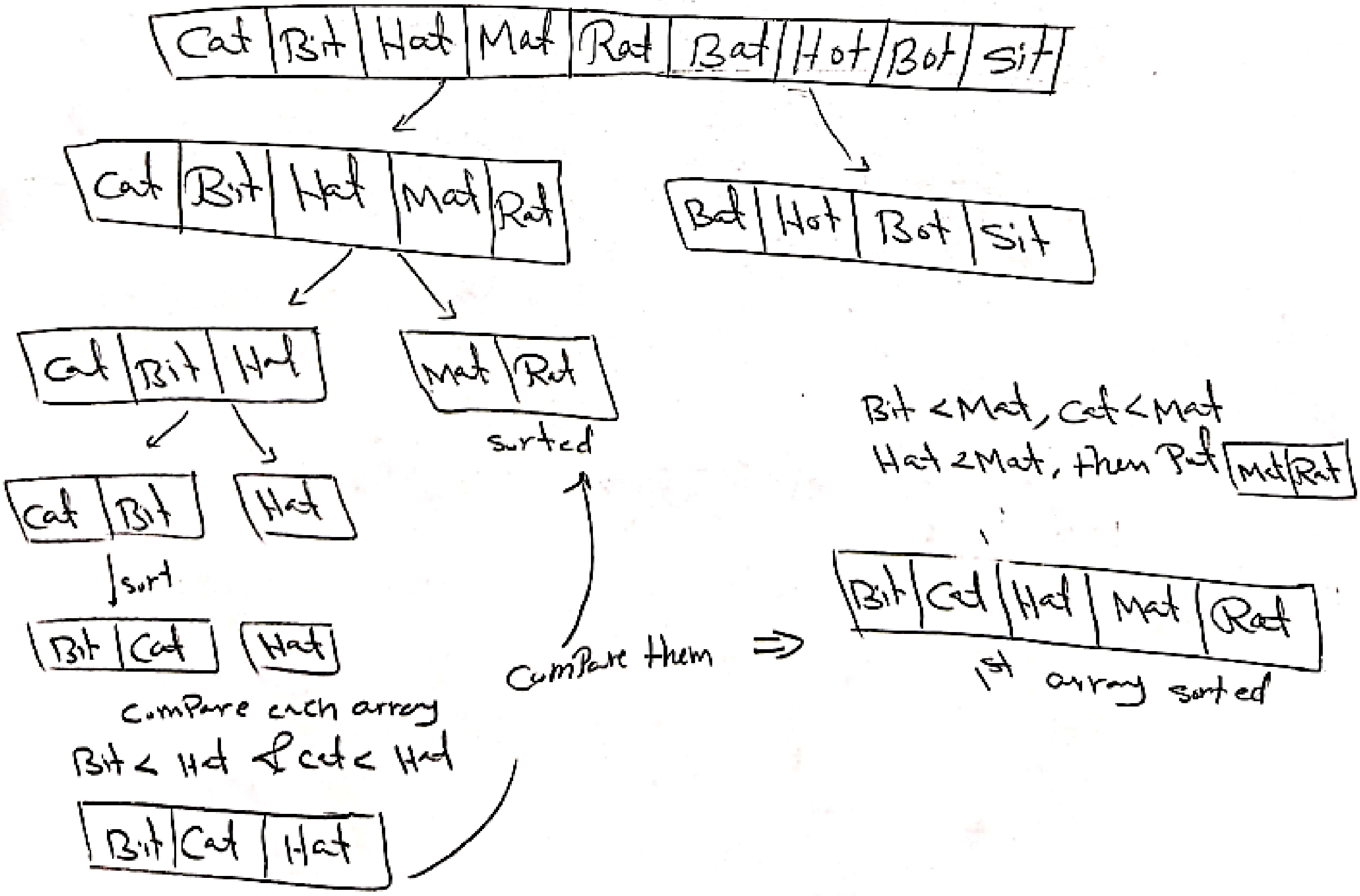
* The final sorted array is

1	2	4	6	8	16	23	30	34	37	41	52	56

---

b)

(b) In ToP down merge sort approach, I divide the array to two arrays and sort each array then Put the values of two arrays in the main arry by comParing each element in arrays one by one and Put the minimum in the main array



Bat	Hot	Bot	Sit

2nd array

Bat	Hot

sorted

Bot	Sit

sorted

Compare two arrays

Bat < Bot, Bot < Hot, Hot < Sit

Bat	Bot	Hot	Sit

2nd array sorted

0	1	2	3	4
Bit	Cat	Hat	Mat	Rat

let i count for 1st array

0	1	2	3
Bat	Bot	Hot	Sit

let j count for 2nd array

i = 0 & j = 0

Bat < Bit so j ++

i = 0 & j = 1

Bit < Bot so i ++

i = 1 & j = 1

Bot < Cat so j ++

i = 1 & j = 2

Cat < Hot so i ++

i = 2 & j = 2

Hat < Hot so i ++

i = 3 & j = 2

Hot < Mat so j ++

i = 3 & j = 3

Mat < Sit so i ++

i = 4 & j = 3

Rat < Sit finish

Bat	Bit	Bot	Cat	Hat	Hot	Mat	Rat	Sit

sorted array

c)

The best sort for this problem is quick sort, by using three elements, if we use 3 way partition which having the elements less than pivot in the left side , large element at the right and elements equal to the pivot beside each other's.which provide linear time which means we will have time complexity O(n).

Question(3):

a)

This occur if the elements already sorted in ascending or descending order as it will split the array to one element (pivot) and the rest of the elements in the other have, and we will keep doing that as it's sorted already. **OR** array with repetitions

---

b)

If we change the order of the elements to be random way, then select the first element as pivot, we will have high probability that the array is not in the worst case especially for array with big data So the best and average case will be **O(n log n)** as usual

---

d)

This occur if the array is already sorted so we will loop for each element(**n-1**) from **0 to n** so the total complexity will be $\boldsymbol{O}(\boldsymbol{n}^2)$

For example if we use the first element as the pivot for numbers [1, 2, 3, 4]

1 2, 3, 4

1 **2** 3, 4

1 **2** **3** 4

1 **2** **3** **4**

And the same if we use the last element as the pivot

1, 2, 3 **4**

1, 2 **3** **4**

1, **2** **3** **4**

1 **2** **3** **4**