Safa Erişti

Sec03

CS 351 – Homework 2

Answer 1.

a)
On IBM 3380, we have 20 blocks of 2400 bytes on each track. The question gives that it takes 0.5 times the read time to process one block of data. This means it also takes 0.5 times the read time to process one track. To illustrate;

I/O	Cpu
Read 1st	-
Read 2nd	Process 1
Read 2nd	-
Read 3rd	Process 2
Read 3rd	-
Read 4th	Process 3
Read 4th	-
Read 5th	Process 4

First Buffer	Second Buffer
-	-
1	-
1	-
1	2
1	2
3	2
3	2
3	4

In this example I/O time is larger than CPU time and since there is overlapping. We only need to calculate the I/O time.

The time it takes to read one track is; 20*ebt (neglecting the s+r)

We have b number of blocks, it means it covers b/20 number of tracks. So the total time is: b/20 * [20 * ebt] = b * ebt

b)

In this case it takes 1.5 times the read time to process one block of data. Meaning that it takes 1.5 times the read time to process one track.

In this scenario, CPU time is larger than I/O time. Since we do overlapping, we can ignore I/O time. So the total time is;

$$b/20 *[3/2 * 20 * ebt] = 1.5 * b * ebt$$

Answer 2.

To find the exact number, I will use IBM 3380's specifications. To find the time for reorganization the formula is; Ty = (b+n/Bfr) * ebt.

For every 3 record we add we delete one record. So it means that when we have 150.000 record, we will have 25000 records marked as deleted.

Bfr = 2400/400 = 6.

If we plug in the numbers we get Ty = (175,000/6 + 150,000/6)* 0.84 = 45,500ms

Sec03
Before reorganizing, we had 175,000, so It takes (175,000/6) * 0.84/2 = 12250ms to fetch

After reorganizing we have 150,000 records. So it takes (150,000/6) * 0.84/2 = 105000ms to fetch

Answer 3.

We are asked to print total population of a town within a county, total population of a county within a city and total population of a city.

First, we open 2-dimensional array in the memory and We start processing the file, for every unique

CITY	COUNTY	TOWN

we add

CITY	COUNTY	TOWN	POPULATION

instance to the array. Whenever we come across a record with same city, county and town, we add its population to the existing population. For example let the first record be Artvin, Şavşat, YavuzKöy, Ahmetler with 200 population, we have

Artvin	Şavşat	Yavuzköy	Ahmetler	200

And let the 10th record be

Artvin	Şavşat	Yavuzköy	Mehmetler	100

Then in the memory we would have

Artvin	Şavşat	Yavuzköy	300

After finishing processing the file we will only deal with the array

Since we have unique entries for every county-town, we just go through the array and whenever we see COUNTY1-TOWN1, we print the population. Now, as we go through the array and print the town populations for a specific county, we also keep the sum of these populations in another variable and print it whenever we finished printing towns of a specific county as the total population of a county within a city.

Last, we need to print the total population of a city. To do this, we need to have another variable for total city populations. In the second part, we kept the populations of the counties. Whenever we finished printing total populations of the counties, we keep this number in the variable and update it (add the population of the new county) every time we finish a new county. This way when we finished printing total populations of the counties of a specific city, we just take the sum of this array and print it as the total population of the city.

We repeat this process for every city and the report would be produced.

If the memory is not big enough to hold this 2-dimensional array, whenever the memory is full we write the array to the disk, and start a new 2-dimensional array. This array can have same entries as the first array. Since we do not know the entries of the first array when we are filling the second.

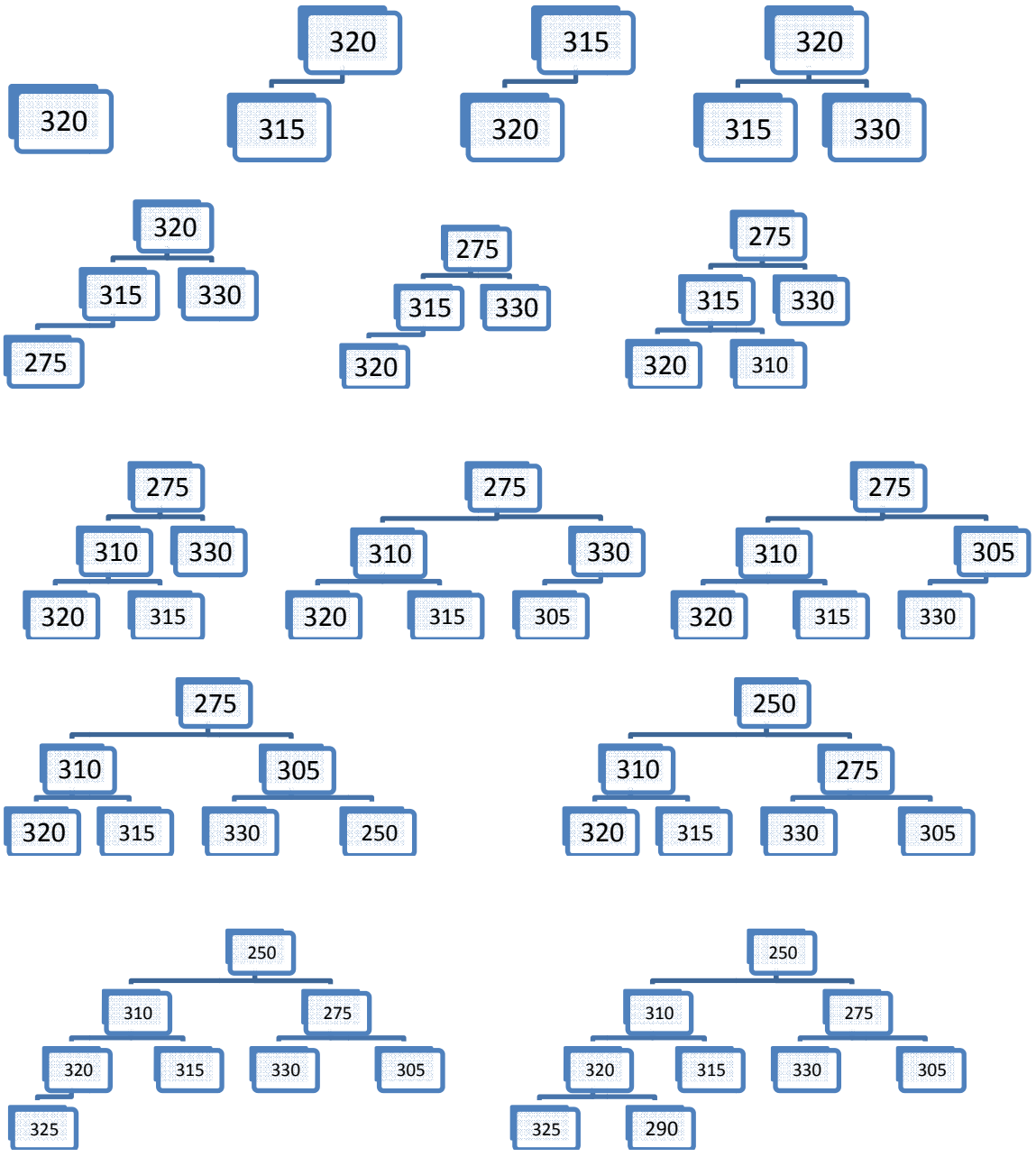
This would require an additional one I/O time for every step. Because we can add the values in the first array, and get the second array from the disk, and continue to add the values and print the result.

Safa Erişti

Sec03

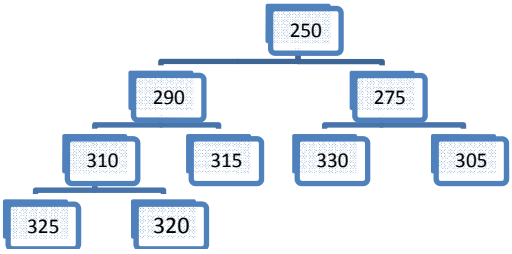
Answer 4.

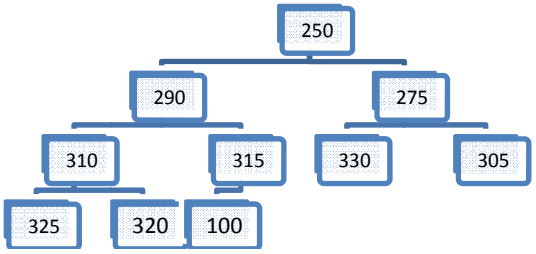
a)

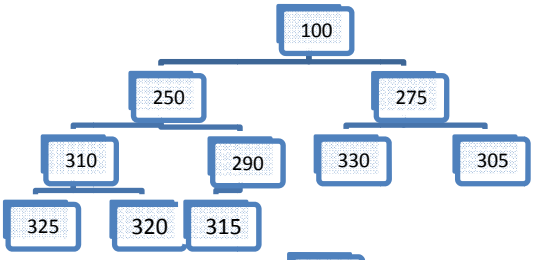


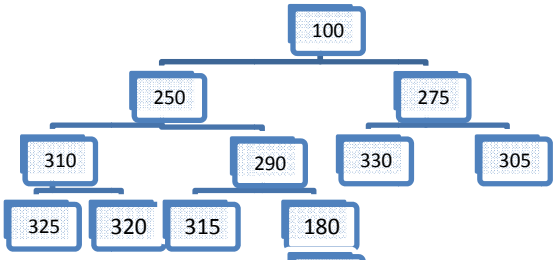
Safa Erişti

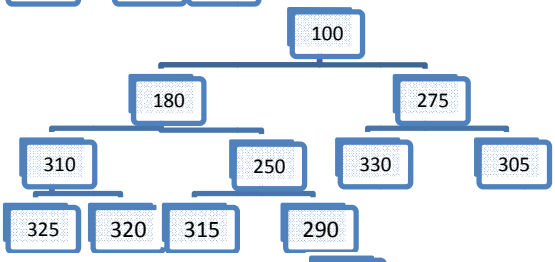
Sec03

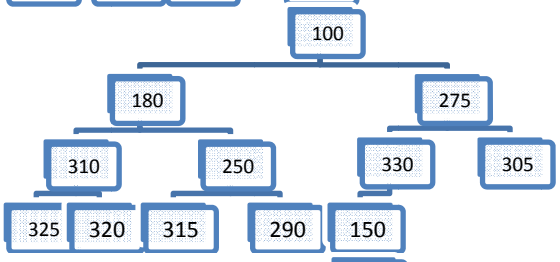


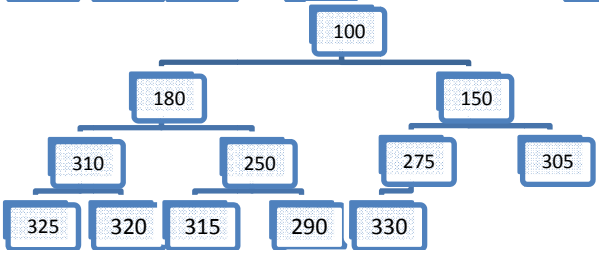


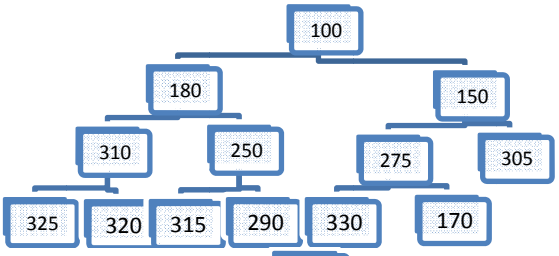


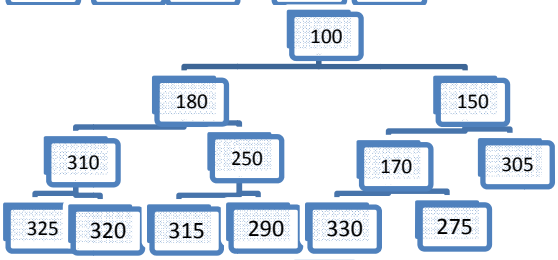


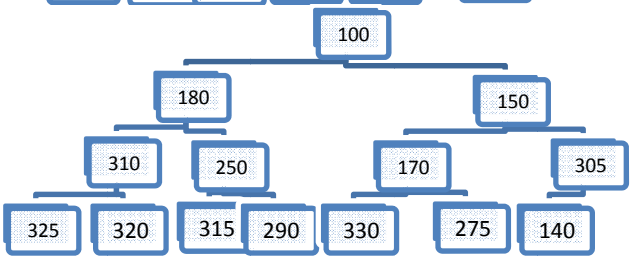


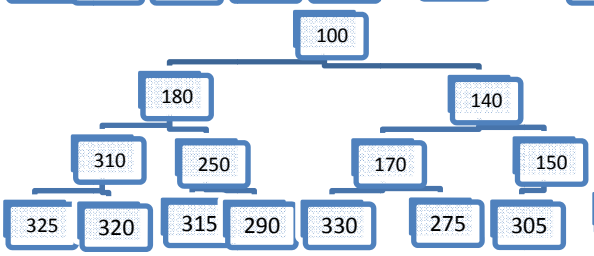


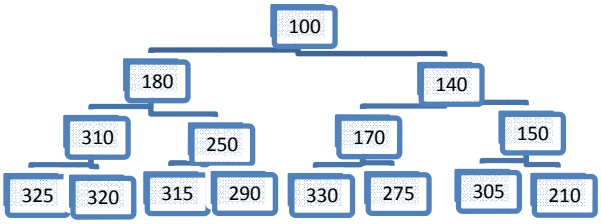


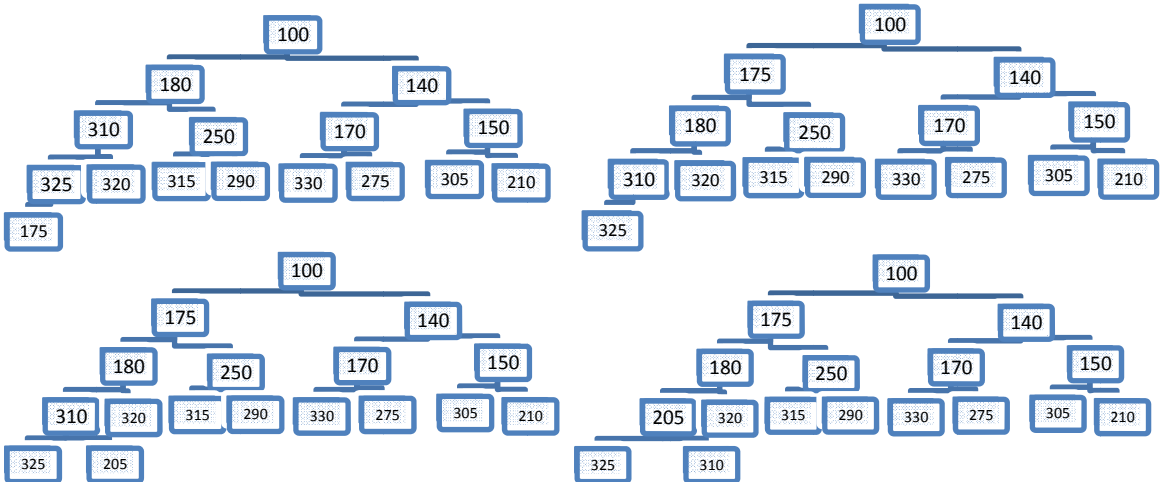




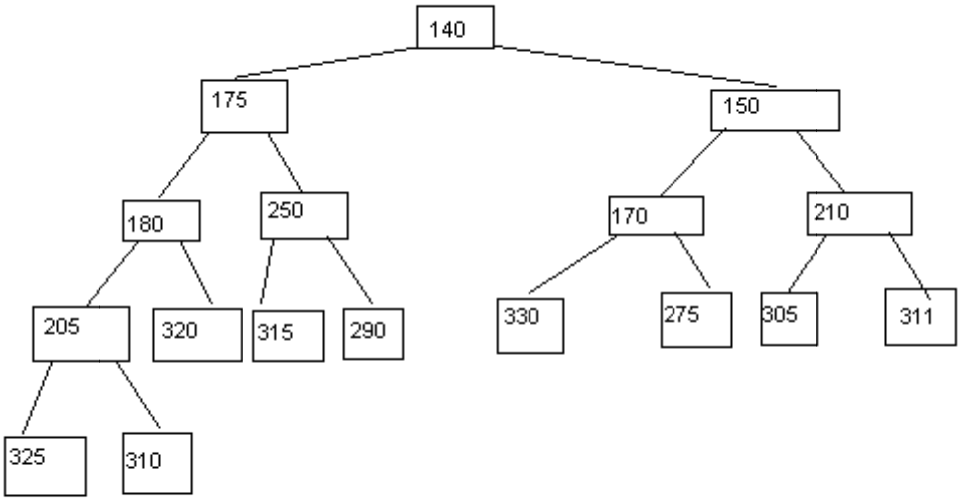








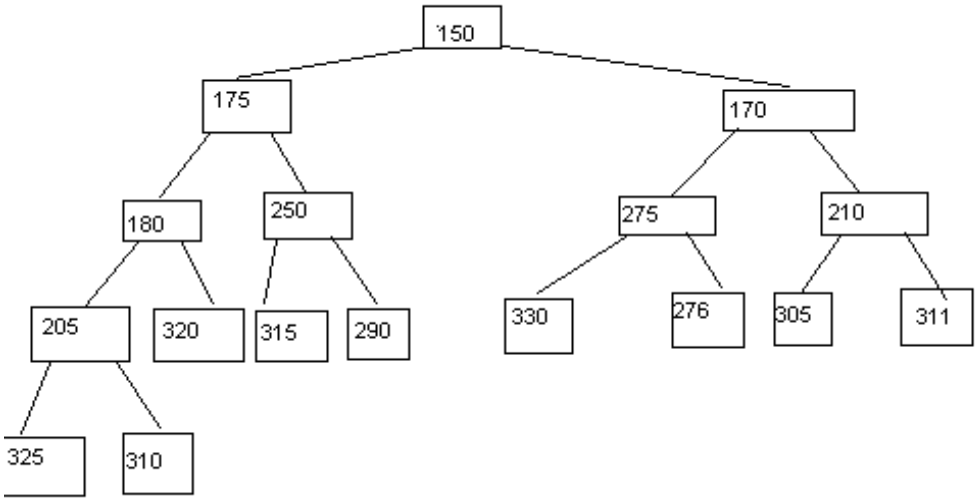
b)Output 100; 311>100; so Insert the newcomer to the root of the current priority queue.
The priority queue becomes;



The output segment;

100																			

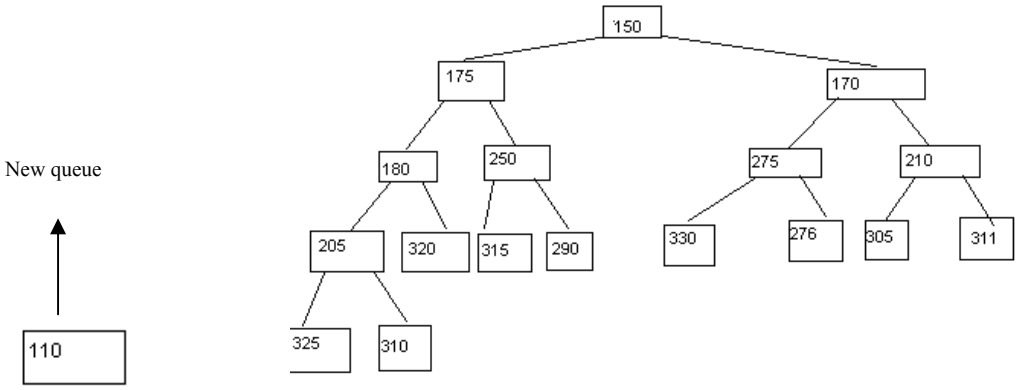
Output 140; 276>140; so Insert the newcomer to the root of the priority queue.
The priority queue becomes;



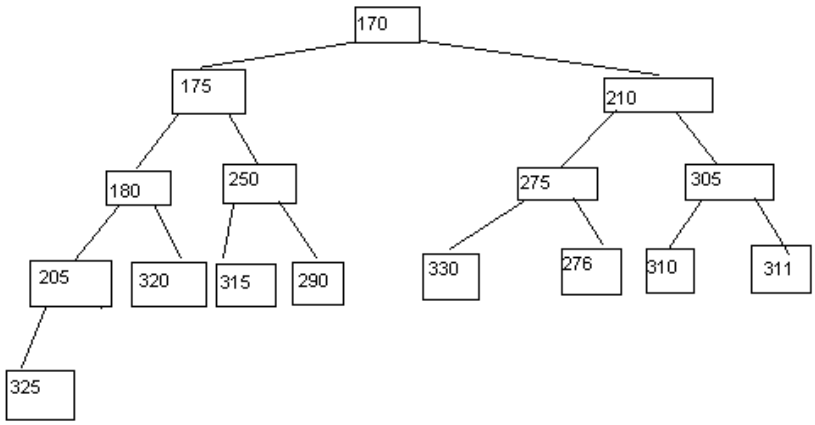
The output segment;

100	140																		

Output 150; 110<150; so newcomer goes to the new priority queue.



becomes;



Safa Erişti

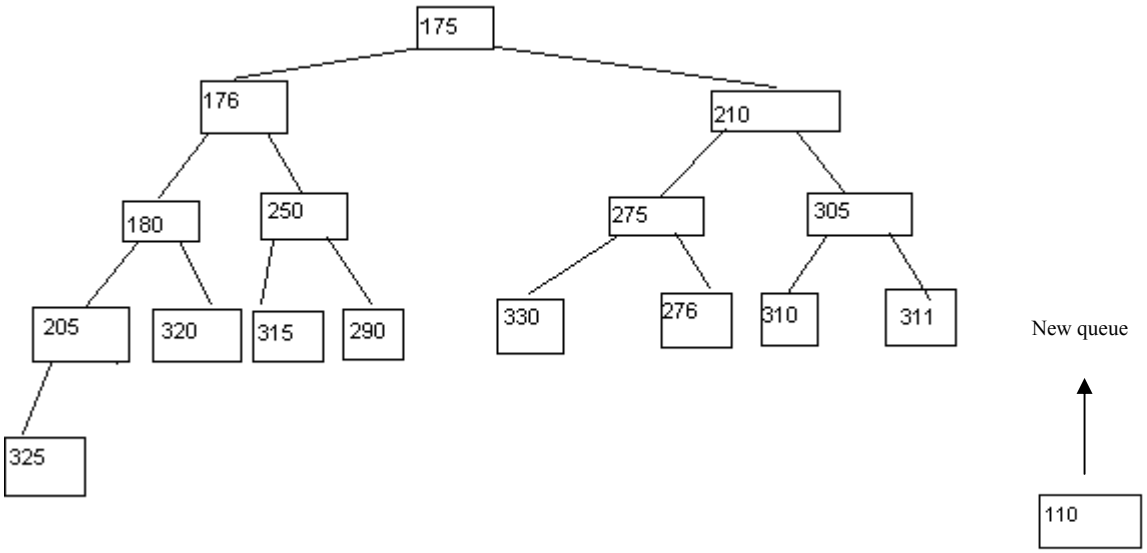
Sec03

The output segment;

100	140	150																	

110

Output 170, 176>170, so insert the new comer to the root of the current priority queue.

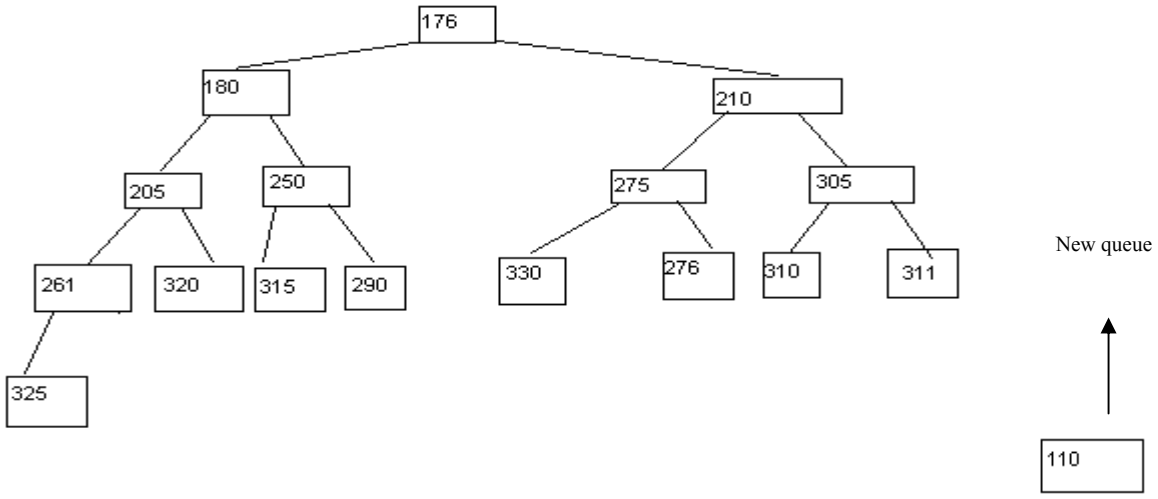


The output segment;

100	140	150	170																

110

Output 175, 261>175, so insert the newcomer to the root of the current priority queue.

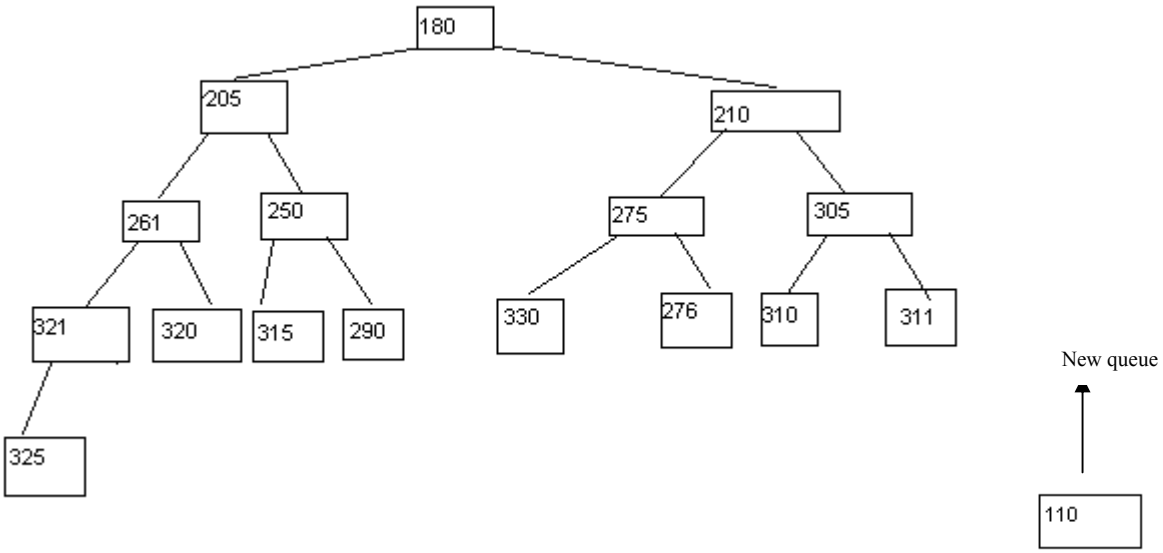


The output segment;

Sec03

100	140	150	170	175														
110																		

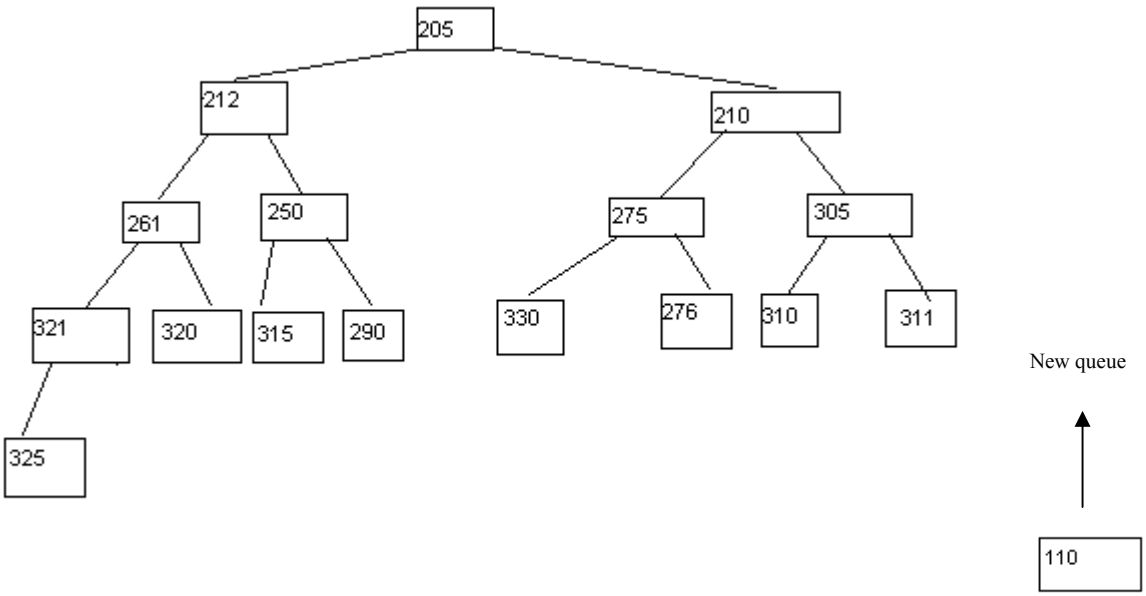
Output 176, 321>176 , so Insert the newcomer to the root of the current priority queue.



The output segment;

100	140	150	170	175	176													
110																		

Output 180, 212 >180, so insert the newcomer to the root of the current priority queue.



Safa Erişti

Sec03

The output segment;

100	140	150	170	175	176	180	205	210	212	250	275	261	275	276	305	310	311	315	320
321	330	325																	
110																			

Answer 5.

a)

We sort the files first. We sort the M1 file with 100-way merge, and sort M2 file with 20- way merge and we open an area on the disk called “common”

Since we have 10Mb of memory,we read 5Mb of M1 file and 5Mb of M2 file to the memory.

Than, we read 5Mb of the M1 file and 5Mb of the M2 file to the memory.

We compare the M1 and M2 and whenever we find duplicate records in we write them to “common” file.

Process goes like;

Take 5MB of M1 Compare with every 5MB of M2,

Take another 5Mb of M1 Compare with every 5Mb of M2

b) Sorting M1 and M2 takes 31 minutes and 70 seconds(Calculation made on the c) part of this question)

For M1 we will go to the disk everytime we want to take a 5Mb piece, so
b * ebt = (5000000)/2400 * 0.84 +(16+8.3) also we have seek and rotational latency time this is for one 5Mb piece and it equals to = 1774.3 ms = 1.7 seconds.

We do this for 200 times, so 200 * 1.7 = 340 seconds

For ever 5Mb of M1 we compare with 5Mb of M2. This means that we take all of M2 to the memory 200 times.

First we calculate bringing M2 to the memory.

b * ebt = (5000000 / 2400) * 0.84 +(24.3) = 1.7 seconds
there are 40 piece of 5Mb pieces, so 1.7 * 40 = 68 seconds

We take M2 200 times, so 68 * 200 = 13600seconds = 226minutes.

Finally there is time to write the common file to the disk.It equals to approximately 4seconds.
So Therefore we have 13600 + 68 + 350 + 70 + 1860 + 4 = 4.43 hours

c)

Without sorting, finding the combined mailing list would take years. So, we definiatly should sort the files first.

Since we have 10Mb of memory, we sort the M1 file with 100-way merge, and sort the M2 file with 20-way merge.

Than, we read 5Mb of the M1 file and 5Mb of the M2 file to the memory.
First we write 5Mb of M1 that we took to memory, to the another area called ”union” and then we start to compare the two 5Mb pieces. Whenever we find nonduplicate records in M2 we write them to union file.
Process goes like;
Take 5MB of M1 Compare with every 5MB of M2,
Take another 5Mb of M1 Compare with every 5Mb of M2

d) First, sorting M1 and M2;

$4b * ebt + 2 * 100^{2} * 24.3 =$ 31minutes for M1
It takes 70 seconds to sort M2.

Now for comparisons

For M1 we will go to the disk everytime we want to take a 5Mb piece, so
$b * ebt = (5000000)/2400 * 0.84 + (16+8.3)$ also we have seek and rotational latency time this is for one 5Mb piece and it equals to = 1774.3 ms = 1.7 seconds.

We do this for 200 times, so 200 * 1.7 = 340 seconds

For ever 5Mb of M1 we compare with 5Mb of M2. This means that we take all of M2 to the memory 200 times.

First we calculate bringing M2 to the memory.

b * ebt = (5000000 / 2400) * 0.84 +(24.3) = 1.7 seconds
there are 40 piece of 5Mb pieces, so 1.7 * 40 = 68 seconds

We take M2 200 times, so 68 * 200 = 13600seconds = 226minutes

Finally there is time to write the union file to the disk.
It takes approximately 35 seconds.

Therefore we have 13600 + 68 + 350 + 70 + 1860 + 35= 4.44 hours