CS351, Fall 2009, HW2 Solutions by Salim Sarımurat

S. 1.

a.

We use track-sized buffers and suppose that processing one block of data takes 0.5 times the read time.

We read in one track, and then we begin processing it while we read in the second track. Since the time for CPU processing is less than I/O time, we are finished with the first buffer when we are in a position to read the third track. So, at the next revolution, we read in the third track and process the second track. Then we read the fourth track and process the third track.

So, while we are processing (Z) th track, we will be reading the next (Z+1) th track and continue processing until the last track of the file is reached.

For this reason, total time is limited to $b \times ebt$.

b.

We use track-sized buffers and suppose that processing one block of data takes 1.5 times the read time.

We read in one track, and then we begin processing it while we read in the second track. We are not yet finished with the first buffer when we are in a position to read the third track because the time for CPU processing is 2 times than I/O time. So, we do not get any new track for one revolution till we finish processing the first track and start processing the second track. Then we read in the third track while finishing the processing of the second track. Then we read in the fourth track and begin processing the third track.

So, while we are finishing the processing of the (Z) th track, we will be reading the next (Z+1) th track. For the next revolution, we will be working on processing the (Z+1) th track so we do no reading. Then we will read (Z+2) th track and start processing the (Z+1) th track. This will continue until the last track of the file is reached.

For this reason, total time is limited to $\frac{3}{2} b \times ebt$.

S. 2.

For deleting 1 record for every 3 records added until the total number of records is 150,000, we need to do **z** number of deletions for **3z** number of insertions on the file.

$$100{,}000\,records - z + 3z = 150{,}000\,records$$

$$z = 25{,}000\,records$$

Finding **Z** as 25,000 records; we say that there are 25,000 records are marked as deleted and 75,000 records are added when we have 150,000 records in the file.

Assuming the block size (B) is 2400 bytes, the blocking factor is $Bfr = \frac{2{,}400}{400} = 6$. Before reorganization, we have $\frac{175{,}000}{6}$ blocks and after reorganization we will have $\frac{150{,}000}{6}$ records in the file.

$$T_Y = \frac{175{,}000}{6} \times 0.84 + \frac{150{,}000}{6} \times 0.84 = 45{,}500 m\sec = 45.5 \sec$$

So, it takes 45.5 second to reorganize this file assuming $ebt = 0.84 m\sec$.

Before reorganization process of this pile file, finding a record time is;

$$T_F = \frac{\left(\frac{175{,}000}{6}\right)}{2} \times 0.84 = 12{,}250 m\sec = 12.25 \sec$$

After reorganization process of this pile file, finding a record time is;

$$T_F = \frac{\left(\frac{150{,}000}{6}\right)}{2} \times 0.84 = 10{,}500 m\sec = 10.5 \sec$$

S. 3.

Since we are expected to report this pile file in an ordered format, we will first sort it. We will accept the files as numbers and use heaps. Assuming the memory is capable of storing this file as a whole, using heaps is the most efficient way to sort.

The way of approaching records while sorting is as following. If we have a record as;

“City3 Country64 Town644 Village 25”

We will mark it as 3-64-644-25 while sorting in order to ease our job.

When we are done with sorting, we will create;

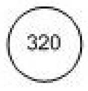
(# of Cities) x (# of Countries) x (# of Towns) amount of variables in the disk and mark them as holding the total population of the towns of every country of cities.

Then we will have the necessary information for reporting. Since these variables are also ordered, we will start listing the report with the given format.

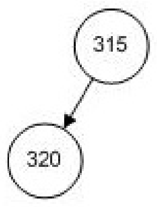
S. 4.

a.

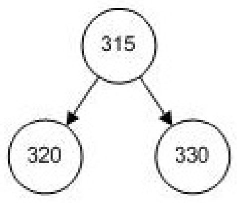
1. **Insert 320**



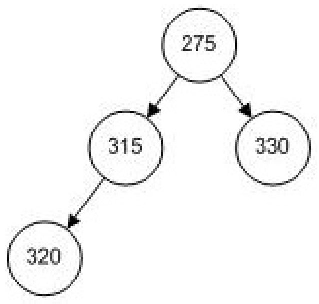
2. **Insert 315**



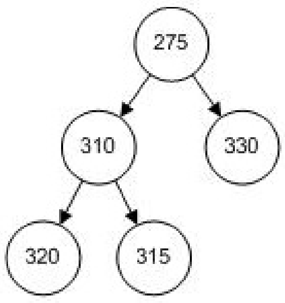
3. **Insert 330**

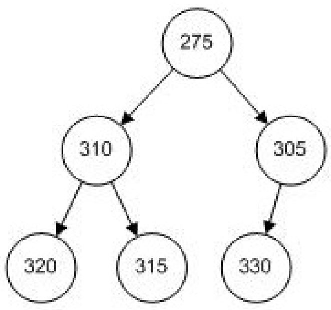


4. **Insert 275**



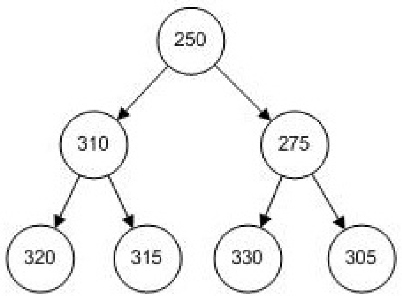
5. **Insert 310**



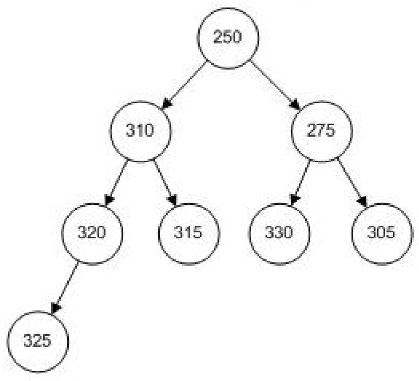


6. **Insert 305**

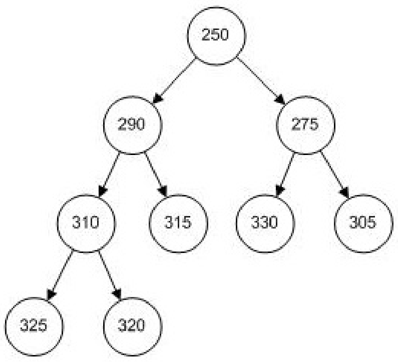
7. **Insert 250**



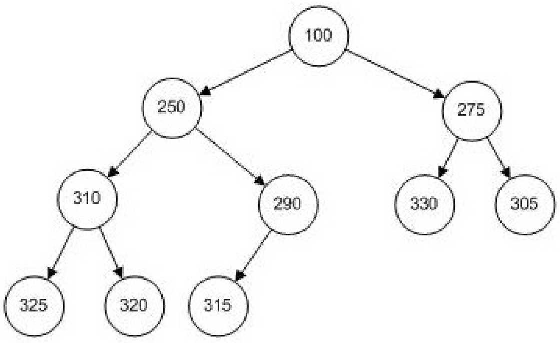
8. **Insert 325**



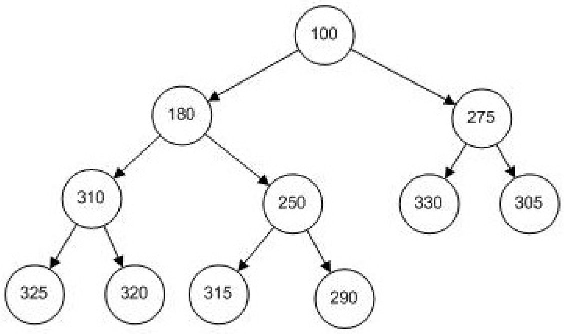
9. **Insert 290**



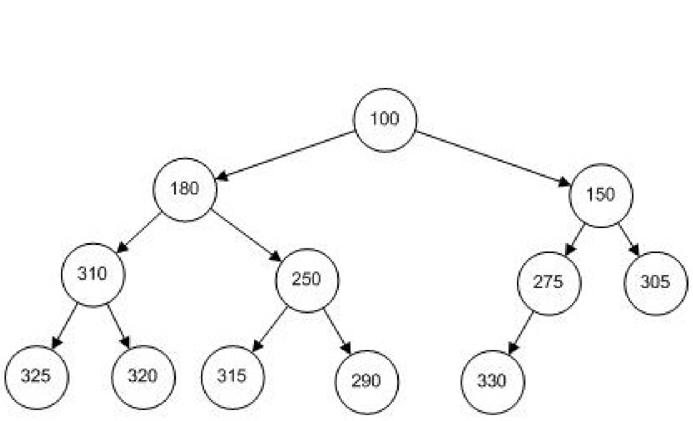
10. <u>Insert 100</u>



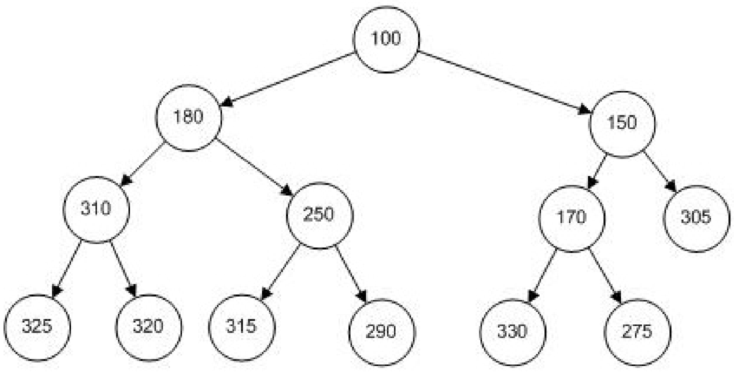
11. <u>Insert 180</u>



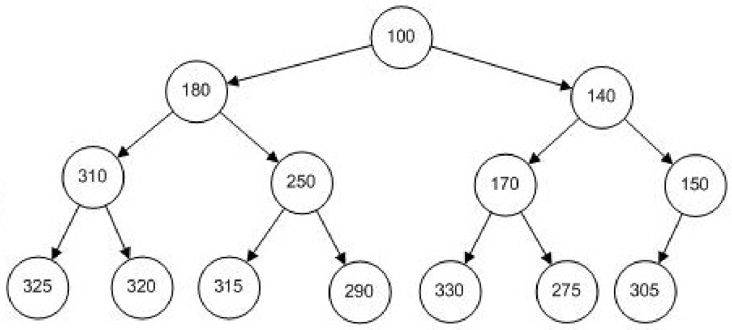
12.<u>Insert 150</u>



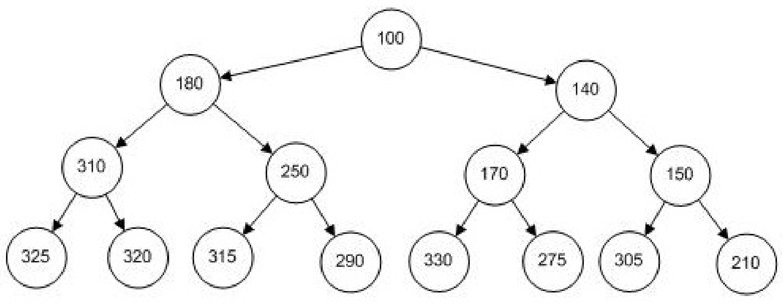
13.Insert 170



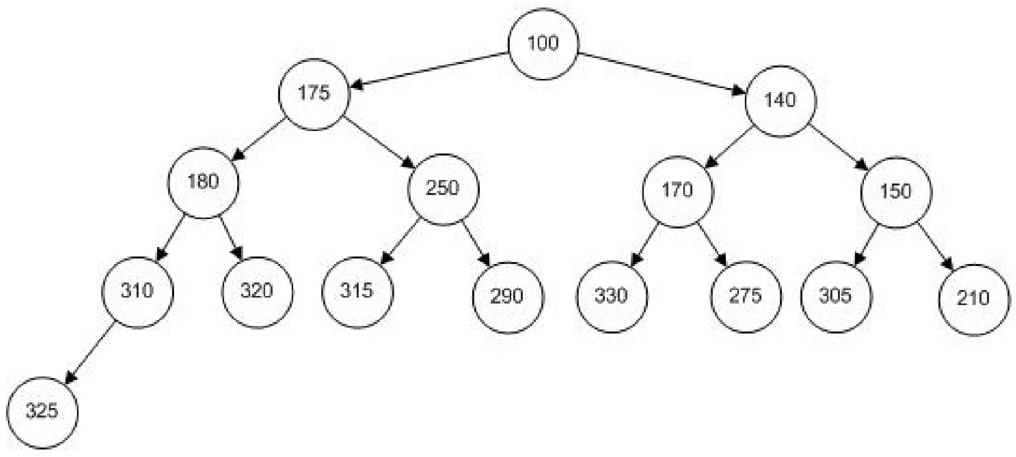
14.Insert 140



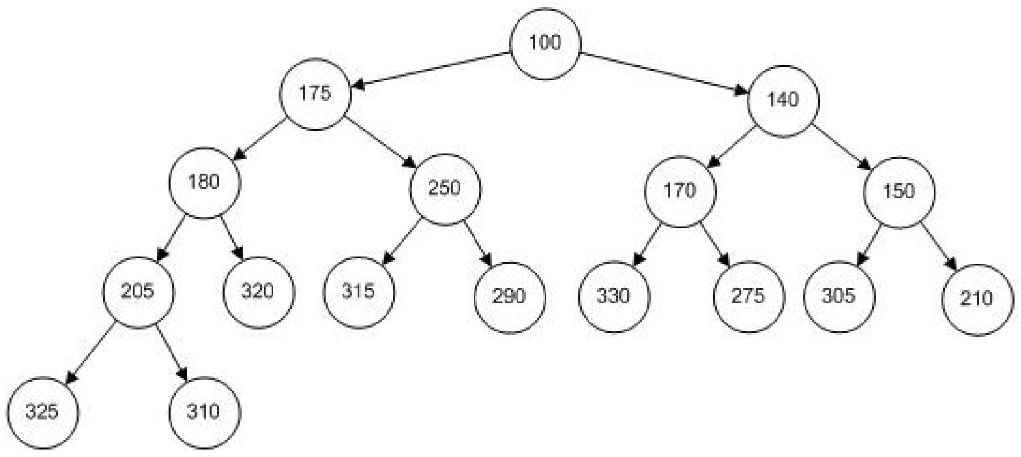
15.Insert 210



16. **<u>Insert 175</u>**

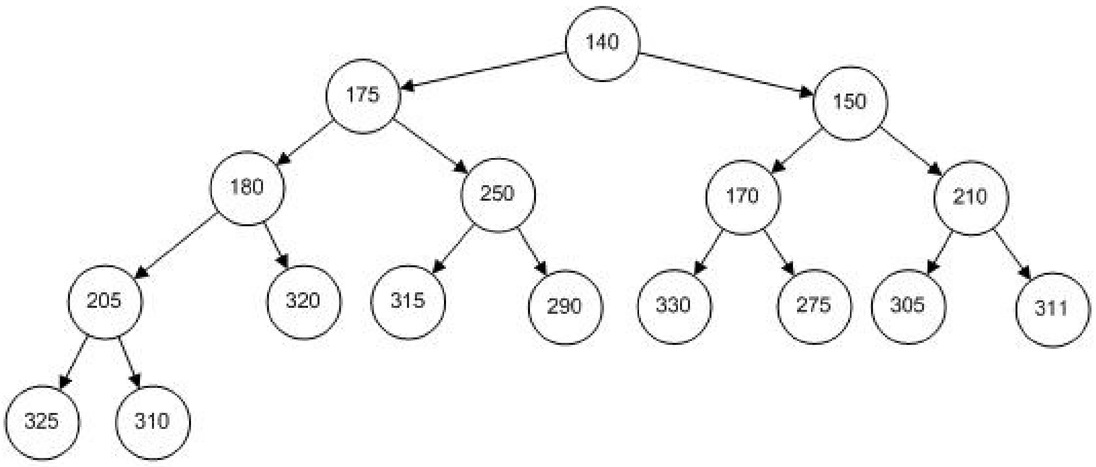


17. **<u>Insert 205</u>**

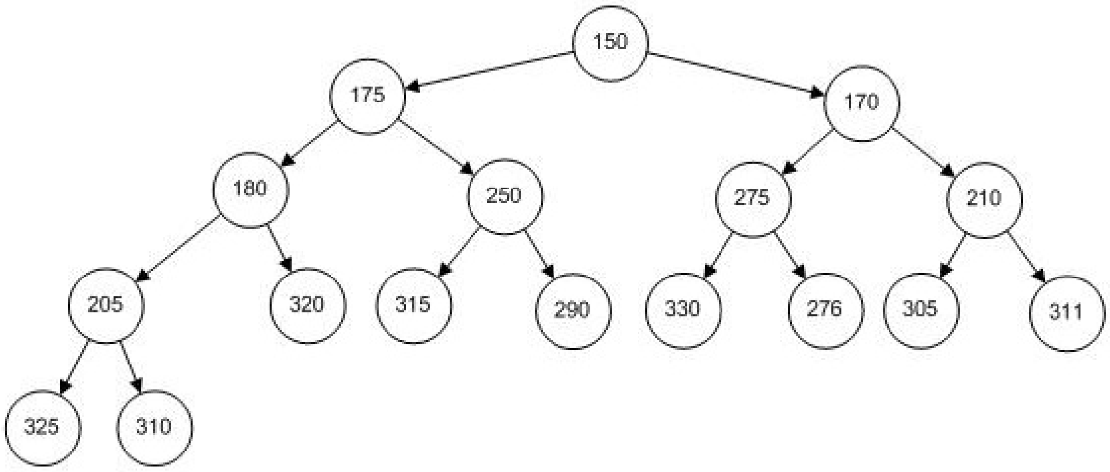


b.

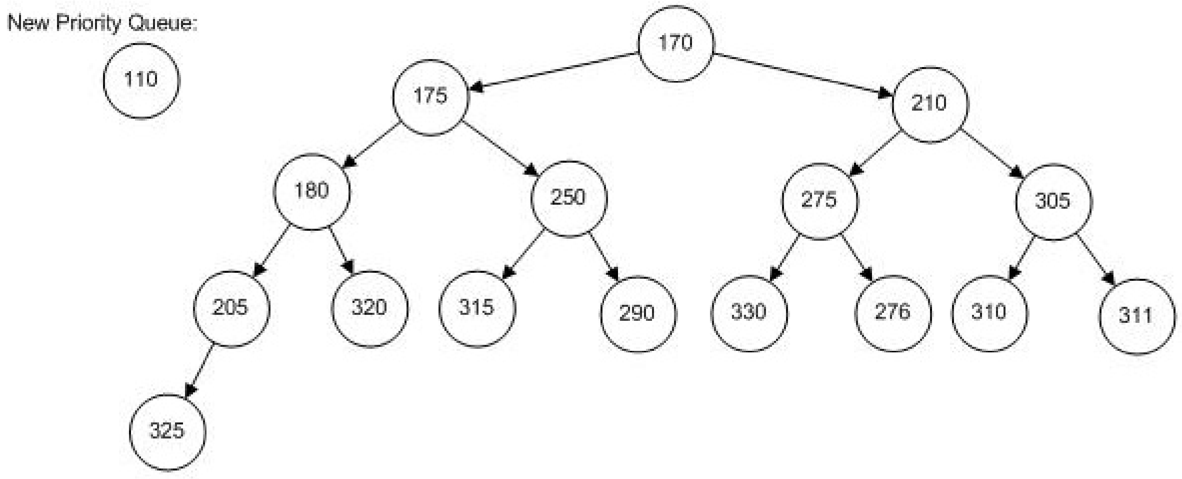
1. **<u>Input 311, Output 100</u>**



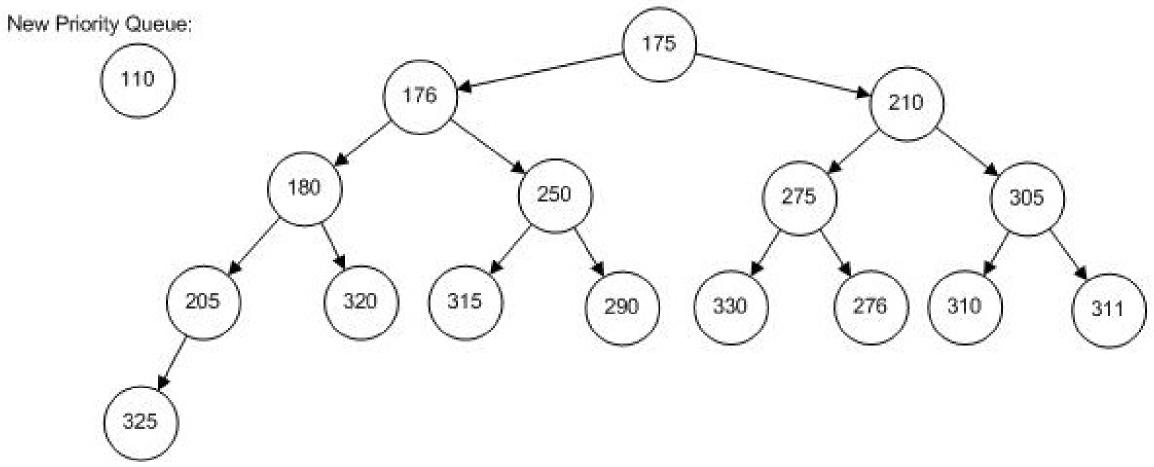
2. **<u>Input 276, Output 140</u>**



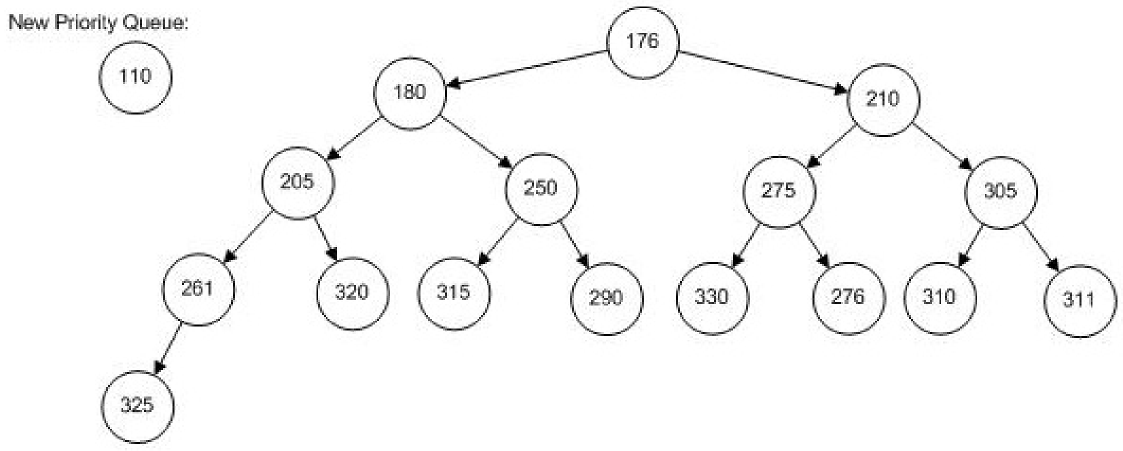
3. **<u>Input 110, Output 150</u>**



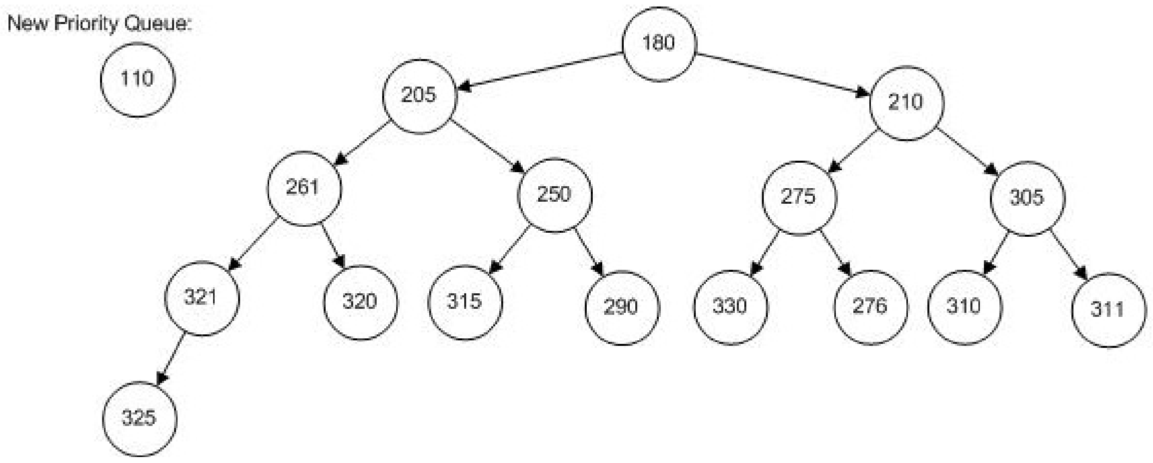
4. **Input 176, Output 170**



5. **Input 261, Output 175**

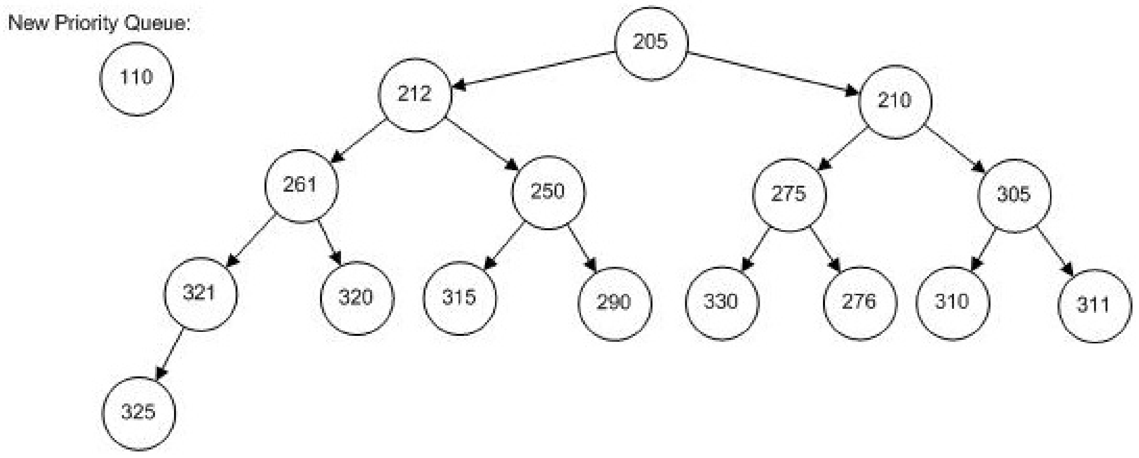


6. **Input 321, Output 176**



---

7. **Input 212, Output 180**



<u>Unit Strings</u>

Old Queue: 205, 210, 212, 250, 261, 275, 276, 290, 305, 310, 311, 315, 320, 321, 325, 330

New Queue: 110

Outputs: 100, 140, 150, 170, 175, 176, 180, 205, 210, 212, 250, 261, 275, 276, 290, 305, 310, 311, 315, 320, 321, 325, 330

S. 5.

Firstly, we should sort the files first. Since the memory size is limited to 10 MB, the most efficient way of sorting the files is using n-way sort method.

The formula provided to sort a file using the n-way sort is the following;

$$T_{Merge} = (2 \times b \times ebt) + [\log_P(nsg)] \times [2 \times P \times (nsg) \times (s + r) + 2 \times b \times ebt]$$

This includes the time to sort the segments first, and the time for log base P of nsg passes. If we can get rid of this logarithm, we will need to consider only one merging pass. So we should make the logarithm base ***P*** and ***nsg*** the same. By doing this, formula becomes;

$$T_{Merge} = 4 \times b \times ebt + 2 \times P \times nsg \times (r + s)$$

and the time for sorting becomes cheap. We will use this logic for both of the files.

Secondly, when we have sorted the files, we will be able to compare them easily. We will divide the memory into two pieces and get 5 MB segments of the files into the memory to compare. Since we have sorted them, by just reading once, we are able to compare and gather the common records.

For example, after we have sorted the files we will read the first 5 MB segments of them into the memory. If the last record of the segment coming from the M1 file is not in the M2 file, we will continue reading 5 MB segments of the M2 file until we reach the last record of the M1 file. Whenever we reach to the last record of the segment coming from the M1 file, we will read another segment from M1 file and continue comparing it with the segments coming from M2 file. So, in this way, we will need to read the files once only.

Thirdly, we will need to write the resulting records into a file. The only difference between the **a)** and **c)** parts of this question is the write phase. For part a) we will write 1 million duplicate records back to the file. For part c) we will write 11 million non-duplicate records back to the file.

a.

1. We will sort the M1 magazine subscription file using the only disk drive that we have. Since we have 10 million records, a 100 way merge sort on 100 segments of 10 megabytes will do the job.
2. We will sort the M2 magazine subscription file using the only disk drive that we have. Since we have 2 million records, a 20 way merge sort on 20 segments of 10 megabytes will do the job.
3. Since M2 file is shorter, we will read that file into the memory and compare it with M1 file. Until the last record of the M2 file, we will do the following process repeatedly;
    a. Read in 5 megabytes of the sorted M2 file into memory.
    b. Read in 5 megabytes of the M1 file up to the record coming after the last record of M2 file in memory.
    c. Compare the records.
        i. If they are **common**, write it to the output file.
        ii. If not, continue with the following 5 MB segment of M2 file.
4. Write the **duplicate 1 million records** into a file.

b.

We know the formula for calculating the time for sorting with one disk drive n-way merge as;

So, sorting M1 file using 100-way merge costs;

$$T_{Merge} = 4 \times \frac{10{,}000{,}000 records \times 100 bytes}{2400 bytes} \times 0.84ms + 2 \times 100 \times 100 \times (16ms + 8.3ms)$$

$$T_{Merge} = 1{,}886{,}000ms = 31{,}4\min$$

Sorting M2 file using 20-way merge costs;

$$T_{Merge} = 4 \times \frac{2{,}000{,}000 records \times 100 bytes}{2400 bytes} \times 0.84ms + 2 \times 20 \times 20 \times (16ms + 8.3ms)$$

$$T_{Merge} = 4{,}990.\overline{6}ms = 5\min$$

Then we will calculate the time necessary for reading both files. Because we need to compare them, it is necessary to read both files entirely. We will ignore the seek and rotational latency times because the number of segments in these files are less.

So, time to read M1 file;

$$T_F = \frac{10{,}000{,}000 records \times 100 bytes}{2400 bytes} \times 0.84 ms = 350{,}000 ms = 350 \sec$$

And time to read M2 file;

$$T_F = \frac{2{,}000{,}000 records \times 100 bytes}{2400 bytes} \times 0.84 ms = 70{,}000 ms = 70 \sec$$

Finally, we will need to calculate the time needed to write these records into the file. Since we have 1 million common records, we will need to calculate the time needed to write these 1 million records into a file.

$$T_{Write} = \frac{1{,}000{,}000 records \times 100 bytes}{2400 bytes} \times 0.84 ms = 70{,}000 ms = 35 \sec$$

So, total time for this whole process is;

T = (sorting M1) + (sorting M2) + (reading M1) + (reading M2) + (writing 1 million records)

T = 31.4 min + 5 min + 350 sec + 70 sec+ 35 sec

T = 44 min

c.

Here we will write one copy of the common records. If two common records came across in the memory, one of them will be written to the file and the other one will be ignored. In this way, we will be able to get the 11 million non-duplicate records.

1. We will sort the M1 magazine subscription file using the only disk drive that we have. Since we have 10 million records, a 100 way merge sort on 100 segments of 10 megabytes will do the job.
2. We will sort the M2 magazine subscription file using the only disk drive that we have. Since we have 2 million records, a 20 way merge sort on 20 segments of 10 megabytes will do the job.
3. Since M2 file is shorter, we will read that file into the memory and compare it with M1 file. Until the last record of the M2 file, we will do the following process repeatedly;
    a. Read in 5 megabytes of the sorted M2 file into memory.

b. Read in 5 megabytes of the M1 file up to the record coming after the last record of M2 file in memory.

c. Compare the records.

   i. If they are **not common**, write it to the output file.

   ii. If not, continue with the following 5 MB segment of M2 file.

4. Write the **non-duplicate 11 million records** into a file.

d.

We already have calculated the time required for most of the steps of the algorithm above. The only thing that we need to calculate is the time for writing the 11 million records back to the file.

$$T_{Write} = \frac{11{,}000{,}000 records \times 100 bytes}{2400 bytes} \times 0.84 ms = 70{,}000 ms = 385 \sec$$

So, total time for this whole process is;

T = (sort M1) + (sort M2) + (read M1) + (read M2) + (write 11 million records)

T = 31.4 min + 5 min + 350 sec + 70 sec+ 385 sec

T = 49.8 min