1.

1)

We use the primary index as an example. Primary index is an ordered file whose records are of fixed length with two fields. The first field is the search key A, and the second field is a pointer to a disk block. The index file is sorted based on the search key A. For the binary search, the number of blocks accesses is $log_2 S$. By using the index file, the number of blocks accesses is $log_2 C + 1$. Since typically C < S, then using index is more efficient.

2)

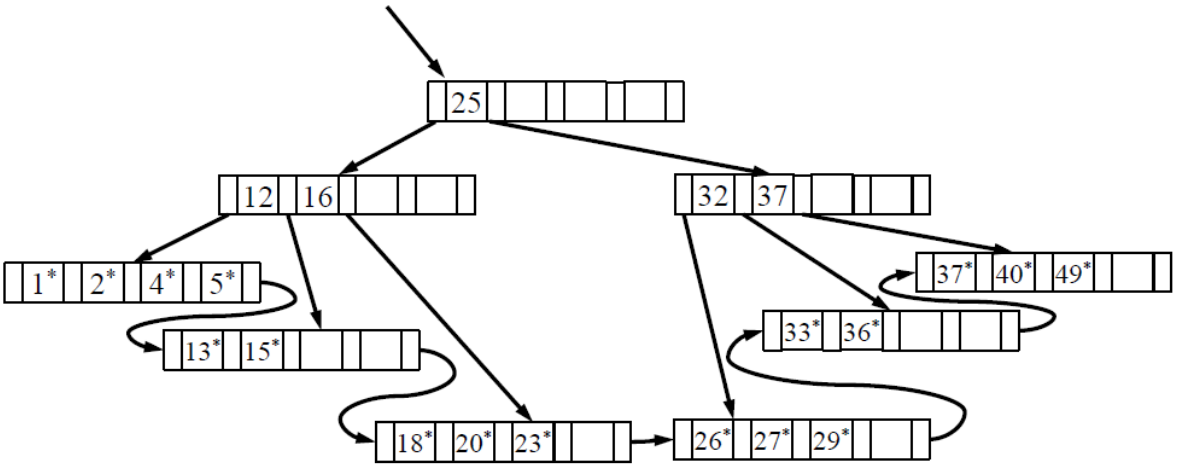
Yes. Consider the current global depth is d ($d > 3$), we have 2^d buckets. Among these buckets, only two buckets have entries. One bucket is for entries of value equal to $k \times 2^{d-3}$ and the other is for entries of value equal to $k \times 2^{d-1}$. Suppose the second bucket only has one entry. Thus, the deletion of this entry will make the second bucket empty. Then we can halve the directory 2 times and decrease the global depth by 2.

3)

Say there are M (M>5) buckets and the overflow pages always happen in the first bucket. Say overflow happens 4 times and the given overflow page threshold of second strategy is 5. Then the first splitting strategy will have M+4 buckets while the second strategy only has M buckets.

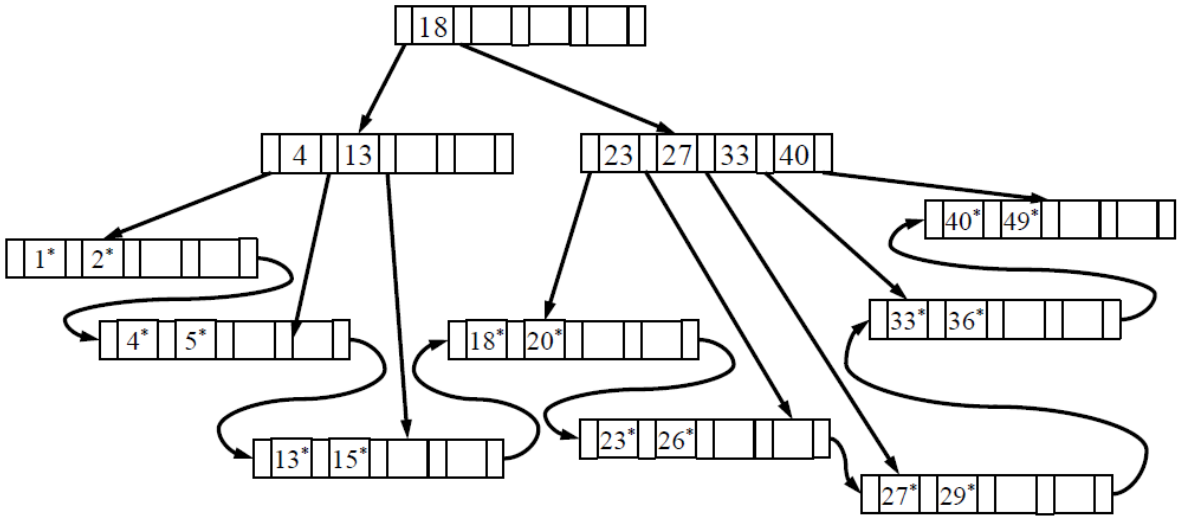
2.

1)



2)

According to the definition of B+-tree, the biggest B+-tree for the same set of index entries will be the tree where every node is just “half full”. Note that the answer is not unique.



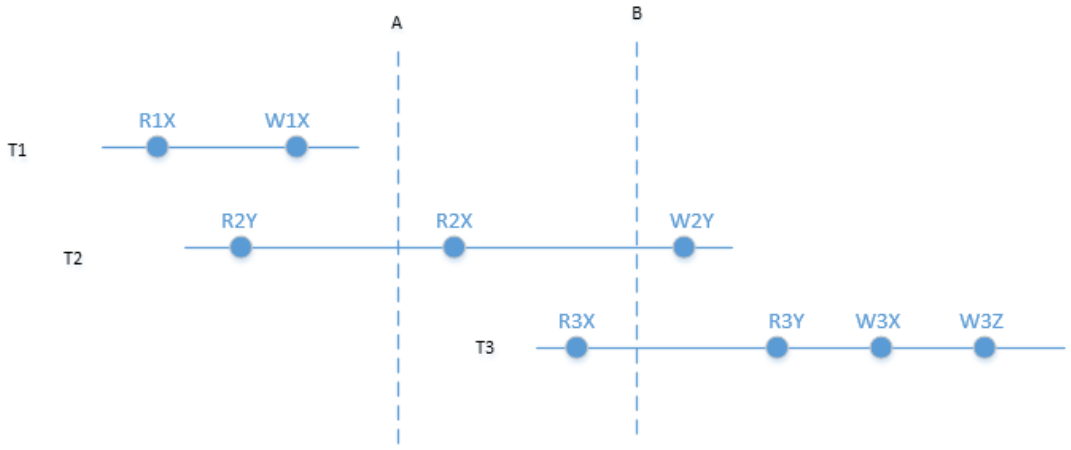
3.

Clustered B+ tree on (R.a, R.b) gives the cheapest costs among the given approaches. The reason is

1) clustered B+ tree performs better for range query as the data are sorted
2) it is an index-only plan, therefore we can retrieve the results without accessing the data records

4

1)



T1: redo

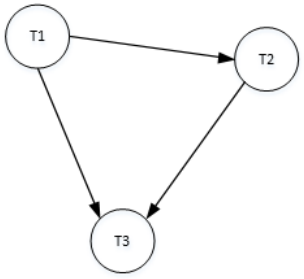
T2, T3: undo

2)

T2, T3: undo

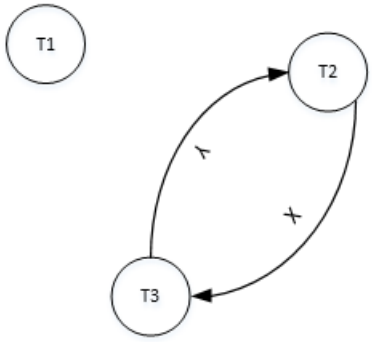
3)

Yes. There is no cycle in its schedule graph:



4) Any schedule whose wait-for graph contains cycles is ok. For example,

T1	T2	T3
write_lock(X)		
Read(X)		
write(X)		
unlock(X)		
		write_lock(X)
		read(X)
	write_lock(Y)	
	read(Y)	
	read_lock(X)	
	wait for X	read_lock(Y)
	wait for X	***wait for Y***
	wait for X	***wait for Y***



5.

The algorithm works like the standard merge sort except we remove the duplicates as soon as we meet them. We start by making $\left\lceil \frac{n}{M-B} \right\rceil$ runs; we fill up the internal memory $\left\lceil \frac{n}{M-B} \right\rceil$ times and sort the recording, removing duplicates. We then repeatedly merge c runs into one longer run until we only have one run, containing K records, where c = $\left\lfloor \frac{M}{B} \right\rfloor - 2$.

I/O analysis:

We consider records of type x_i. In the first phase we read all the n_i records of this type. In phase j there are less than $\frac{\lceil N/(M-B)\rceil}{c^{j-1}}$ runs , where c = $\left\lfloor \frac{M}{B} \right\rfloor - 2$ and we therefore have two cases:

1) $\frac{\lceil N/(M-B)\rceil}{c^{j-1}} \geq n_i$: there are more runs than records of the type x_i, this means that in the worst case we have not removed any duplicates, and therefore all the n_i records contribute to the I/O complexity.

2)$\frac{\lceil N/(M-B)\rceil}{c^{j-1}} < n_i$: There are fewer runs than the original number of x_i. There cannot be more than one record of the type x_i in each run and therefore the record type x_i contributes with no more than the number of runs to the I/O complexity.

The solution to the equation $\frac{\lceil N/(M-B)\rceil}{c^{j-1}} = n_i$ with respect to j gives the number of phases where all n_i records might contribute to the I/O-complexity. The solution is $j = log_c\left(\frac{\left\lceil \frac{n}{M-B} \right\rceil}{n_i}\right) + 1$, and the number of times the record type x_i contributes to the overall I/O-complexity is no more than:

$$n_i\left(log_c\left(\frac{\left\lceil \frac{N}{M-B} \right\rceil}{n_i}\right)+1\right)+\sum_{j=log_c\left(\frac{\left\lceil \frac{N}{M-B} \right\rceil}{n_i}\right)+2}^{log_c\left(\left\lceil \frac{N}{M-B} \right\rceil\right)+1}\frac{\lceil N/(M-B)\rceil}{c^j}$$

Adding together the contributions from each of the k records we get the overall I/O-complexity:

$$\text{IO} \leq \frac{2}{B}\left(N+\sum_{i=1}^{K}\left(n_i\left(log_c\left(\frac{\left\lceil \frac{N}{M-B} \right\rceil}{n_i}\right)+1\right)+\sum_{j=log_c\left(\frac{\left\lceil \frac{N}{M-B} \right\rceil}{n_i}\right)+2}^{log_c\left(\left\lceil \frac{N}{M-B} \right\rceil\right)+1}\frac{\lceil N/(M-B)\rceil}{c^j}\right)\right)$$

Based on this inequality, we can get the I/O complexity as $O(\max\{\frac{N}{B}log_{\frac{M}{B}}\frac{N}{B}-\sum_{i=1}^{K}\frac{N_i}{B}log_{\frac{M}{B}}N_i,\frac{N}{B}\})$. Detail derivation process can refer to the paper titled “A General Lower Bound on the I/O-Complexity of Comparison-based Algorithms”