Sorting 1 trillion integers

Given a set of 1 Trillion integers on hard disk, find the smallest 1 million of them. You can fit at most 1 million integers in memory at a time.

One approach is, take the first 1 million out of 1 trillion and sort the 1 million integers and store it back in the hard disk. In this way carry on the sorting for every group of 1 million integers and store it in the hard drive. Now groups of 1 million integers are sorted up to 1 trillion. Now compare the first element of all the sorted groups the minimum of them is the minimum of the 1 trillion. Store it as the first element in the memory. Next, take the second element from the group from which the smallest element came and then check it with all other groups first element. In this way repeat the procedure until the first 1 million is sorted and stored in the memory.

Is there a more optimal approach that I'm missing?

algorithm sorting

edited Jun 16 '11 at 5:53

asked May 27 '11 at 18:11

dreamer
162 4 15

3 Interview question? – Adam Byrtek May 27 '11 at 18:13

3 Sounds like homework to me... – Limey May 27 '11 at 18:14

4 European or American trillion? Either way, though, use most of the million integers-worth of memory to memory-map three sections of the file at a time, and the rest as stack space to run quickselect. – Steve Jessop May 27 '11 at 18:14

3 @Chuck: American trillion is 10^12. European is 10^18, so you'd have 10^12 sorted chunks after the first step. It was a slightly silly question: for most purposes people use American English for anything computer-related, so it's normally going to be the former. – Steve Jessop May 27 '11 at 18:19

2 If you sort 3 groups of 3 (413)(928)(657)=>(134)(289)(567) and then choose the smallest at index (0..2), you get (134) and will miss the 2. Or did I understand you wrong. – user unknown Jun 23 '11 at 13:15

show **1** more comment

4 Answers

You can do this efficiently in O(n log m) by using a heap. (n = all the numbers, m = the size of the set of numbers you want to find).

Go through the trillion numbers one at a time. For each new number do one of the following.

1. If the heap has < 1 million nodes insert the new number into the heap.
2. If the heap has exactly 1 million nodes and the top node is > than the new number, then pop the top node from the heap, and insert a node with the new number.
3. If neither 1 or 2 are true then toss the number.

After you go through all the trillion entries then the resulting heap will have the 1 million smallest numbers.

Inserting and deleting from the heap is O(log m). The single pass through the heap is n. So, the algorithm is n*log (m)

edited Jun 23 '11 at 1:12

answered May 27 '11 at 18:18


Himadri Choudhury
5,428 4 21 34

3 I wouldn't call this O(n log n), since the n outside and the n inside the log are different numbers. – sverre May 27 '11 at 18:23

don't you have to sort the heap after every insert? – F.C. May 27 '11 at 18:27

2 @F.C. No. After each insert the top of the heap will always be the biggest element. That's a basic property of a heap. You are only checking the top element anyway. – Himadri Choudhury May 27 '11 at 18:28

@sverre: I see your point. Edited the answer to say O(n log m) instead. – Himadri Choudhury May 27 '11 at 18:48

@Scott Urban The edit you offered on this answer changed the meaning (even if it was only to refine it). This is not what edits are for. Wait until you have the reputation point for writing comments, and use that instead. – Pascal Cuoq May 28 '11 at 15:40

show **1** more comment

How large are the integers? If they're just 32-bit values, I would simply make an array of 4 billion 64-bit counters on disk, and upon encountering `x` in the input, increment the counter in position `x`. In general this approach is extremely costly in space, but proportionally the cost is low whenever the range of possible element values is much smaller than the number of items to be sorted, and best of all it's `O(n)` in time.

answered May 27 '11 at 20:07

R..
102k 10 135 322
add a comment

A solution in scala, but not for 1 Trillion elements. With a pointer into a file instead of the List, or multiple small lists, it could be done in this way:

```
def top (n: Int, li: List [Int]) : List[Int] = {

  def updateSofar (sofar: List [Int], el: Int) : List [Int] = {
    // println (el + " - " + sofar)
    if (el < sofar.head)
      (el :: sofar.tail).sortWith (_ > _)
    else sofar
  }

  /* better readable:
    val sofar = li.take (n).sortWith (_ > _)
    val rest = li.drop (n)
    (sofar /: rest) (updateSofar (_, _)) */
  (li.take (n). sortWith (_ > _) /: li.drop (n)) (updateSofar (_, _))
}
```

Take the first million elements. Sort them. Now for every following element, compare it to the biggest in the million. If it is smaller, sort it into the list and drop the old biggest.

answered Jun 23 '11 at 13:28

user unknown
16.5k 5 31 73
add a comment

You can do this even more efficiently using a variant of QuickSort, in O(n) time, where 'n' is the size of the list on disk. (in this case a Trillion)

All you have to do is:

1. Find the one millionth smallest number by partitioning the disk drive several times in increasingly small sections. This takes O(n) time.
2. Take it and the other 999,999 integers that the partitioning has sorted out and put them in RAM. You are done.

The smallest million integers will not be sorted, but they will be the smallest million.

If you then want to sort the smallest million, it will take O(m log m) time, where 'm' is a million in this case.

No cost to space, O(n) time, works with non-integer values. Enjoy. :)

answered Jun 16 '11 at 5:43

heated
1

efficiency is O(n^2) in the worst case – dvhh Jun 17 '11 at 5:40

add a comment

Not the answer you're looking for? Browse other questions tagged algorithm sorting or ask your own question.