Qixiang (David) Zhang
Nai-Jack Li

Compare based sorting

Problem statement

Find k largest elements in an unsorted size-n array with the least number of comparisons possible.

> For example, for n=7 and k=3,
> if the private array (indexed from 1 to 7) contains the values: 3,5,4,7,2,6,1
> then the integer array Best[] (indexed from 0 to 2) should hold: 4,6,2

Our result

In this project, the **sortAlgo.c** uses the framework of insertion sort and the mechanism of binary search. With parameters $n = 10000$, $k = 40$, and $iteration = 1000$, the maximum number of comparisons is `11558` , and the average number of comparisons is `11316.97` .

Our algorithm

First, initialize an array called $Best$ with k number of -1's. Then take one element e from the input array and compare it with the last item of $Best$ array $Best[k-1]$. If smaller, eliminate this choice and move on to the next element. If larger, replace the last item of $Best$ array with this element $Best[k-1] = e$. The items in the $Best$ array are maintained in descending order based on their values. By binary search, find the exact location i to put the element $Best[i] = e$, and shift the rest of the items to the right. The process continues until all elements of the input array have been inserted or eliminated.

`drive.c` gets invoked every time when elements gets compared.

Here is the pseudo code:

```
doalg(n, k, Best)
    Initialize Best array - size k of -1's
    loop i from 1 increment to n
        Compare i with the last item of array Best
        If Best[k-1] = -1 OR Best[k-1] < i then
            Best[k-1] = i
            Use binary search to find where to insert i
            insert i and shift the rest of the elements to the right
    return 1
```

Below is the result of running the algorithm with different n and k:

n	k	Observed WC	Observed AVG
100	10	231	190.51
-	20	343	285.54

n	k	Observed WC	Observed AVG
-	40	512	424.42
1000	10	1247	1167.93
-	20	1494	1386.30
-	40	1991	1823.90
10000	10	10345	10247.75
-	20	10732	10590.82
-	40	11558	11316.97

Analysis for Theoretical Worst Case

Theoretically, the worst case for the number of comparisons occurs when the input array is sorted in ascending order. Every element in the input array has to be compared with the smallest item once inside the $Best$ array ($size = 40$) and then goes through the binary search process for a maximum of $ceil(log_2 40) = 6$. Follow this logic, let n be the size of the input array, then the number of total comparisons would be:

$$n[ceil(\log_2 k) + 1]$$

In this case ($n = 10000,\ k = 40$), the total number of comparisons is 70000.

The total permutation of the size-n array is $10000!$, which is 35,658 digits long (a very large number). Then, the probability of the input array that is already sorted in ascending order is $\frac{1}{10000!} \approx 0$. Therefore, even if the algorithm runs for 1000 times, the chance for the worst case to occur is still close to zero.

To validate our hypothesis, we manually create an array with size 10000 and in ascending order. The observed total number of comparisons for the worst case is 69919, which is 81 comparisons short of 70000. This minor inconsistency occurs because we over counted the comparisons take place in the beginning of the process - it takes less than $n[ceil(\log_2 k) + 1]$ when the k size $Best$ array is not fully constructed.

Analysis for Theoretical Expected Worst Case

The expected worst case is the maximum possible number of comparisons we expect to see by running this algorithm on 1000 random input size-n** arrays. The expected worst case comparisons is the sum of the products of the probability and total comparisons of each worst case.

Let P_i be the probability of the worst case, and n be the total number of elements in the input array, n_i be the number of the sub-array which contains i th largest element to the largest element in ascending order. The maximum possible permutation of the input array is $n!$. The maximum possible permutation of the sub-array is $(n - n_i + 1)!$. Thus, the probability P_i for the number of the largest elements already sorted in ascending order in the input array is

$$P_i = \frac{(n - n_i + 1)!}{n!}$$

Let N_i be the number of maximum possible comparisons, we can use the logic from the above section with a little modification. The total number of comparisons for n_i is $n_i(ceil(\log_2 k) + 1)$. The total number of comparisons for the rest of the elements ($n - n_i$) is 1 for each, so $(n - n_i) \times 1$. Therefore, the number of comparisons that the worst cases uses is

$$N_i = n_i[ceil(\log_2 k) + 1] + (n - n_i)$$

The probability of the worst case does not occur in 1000 times is $(1 - P_i)^{1000}$, and the probability for the worst case does occur at least once in 1000 times is $1 - (1 - P_i)^{1000}$. Therefore, the total expected worst case E_{WC} is

$$E_{WC} = \sum_{i=1}^{n} N_i[1 - (1 - P_i)^{1000}]$$

When $n = 10000$ and $k = 40$, the result is `10958.9`. To verify our hypothesis, we ran **main.c** and observed that the actual worst case is 11558. The difference is 5.467% which is negligible.

Analysis for Theoretical Average Case

The average number of comparisons is the sum of the products of the probability and the average number of comparisons for each possible cases. On average, binary search sorting algorithm takes $ceil(log_2 k)$ comparisons for k elements. For every element in the input array, exactly 1 comparison is needed for testing if the element is larger than the smallest item in the $Best$ array. If larger, the element would be inserted into $Best$ for binary search and result in more comparisons.

Let num be an element in the input array, P_{num} be the probability that this element is smaller than the smallest item in the $Best$ array (so the binary search does not apply to this element)

$$P_{num} = P(\text{binary search does not apply to } num)$$

then

$$\text{total average number of comparisons} = \sum_{num=1}^{n} [(ceil(log_2 k) + 1)(1 - P_{num}) + 1 \times P_{num}]$$

To derive P_{num}, let i be the index of num in the input array. Let $p_{num,i}$ be the probability that binary search does not apply to num, which at i th position in the input array. In other words, $p_{num,i}$ is the probability that at least k elements that are larger than num and on the left of the i th position.

Since each elements in the input array are unique and differs by 1. Then there are $n - num$ out of $n - 1$ elements larger than num and there are $i - 1$ elements in front of i th position. Then, n_{front}, the average number of elements in front of i th position that are larger than num,

$$n_{front} = \frac{n - num}{n - 1} \times (i - 1)$$

If n_{front} is larger than k (in other words, there are average more than k large elements in front of i th position), $p_{num,i}$ would be close to 1.

On the other hand, if n_{front} is smaller than k (in other words, the average number of elements is less than k), we assume $p_{num,i}$ and n_{front} are in positive linear correlation. Then,

$$p_{num,i} = \begin{cases} 1 & n_{front} >= k \\ n_{front}/k & n_{front} < k \end{cases}$$

Then, P_{num} is just the sum of $p_{num,i}$ from 1st position to nth position with the normalized factor

$$P_{num} = \sum_{i=1}^{n} (\frac{1}{n} p_{num,i})$$

Finally, let $n = 10000$ and $k = 40$, the theoretical average total number of comparisons: `10907` , which is close to the actual average number: `10590.82` .

Analysis for Observed Worst Case and Average Case and Inconsistencies

In summary, with $n = 10000$ and $k = 40$, the theoretical worst case is 10958.90 vs. the actual observed worst case 11558; the theoretical average case is 10907 vs. the actual observed average case 11316.97. Since the differences are negligible, the observed cases validate the analysis above.

Below is the table in summary of the result number of comparisons of the algorithm on 1000 random arrays:

n	k	Observed Worst	Expected Worst	Observed Avg	Expected Avg
10000	40	11558	10958.90	11316.97	10907.00