XJTLU | Computing

Sortings

Questions

Problem 1. State the running time of merge sort when the input is:

(a) random

(b) sorted

(c) sorted in reverse order

(d) with all elements the same

Problem 2. Sort the integers $\{6, 0, 2, 0, 1, 3, 4, 6, 1, 3, 2\}$ with quick sort.

Problem 3. What is the maximum number of recursive calls that need to be stored in the stack when running (i) merge sort, and (ii) quick sort, on an array with n elements?

Problem 4.

(a) The main loop of the insertion sort uses a linear search to scan through the sorted subarray $A[0 : j - 1]$ to find the insertion point. Since the subarray is already sorted, can we improve the worst-case running time by using binary search to find the insertion point? If so, what would be the improved worst-case running time?

(b) Would using binary search to find the insertion point in insertion sort be a good idea in practice? Why or why not? Take into account the kids of data structure on which you would use in insertion sort.

Problem 5. For each of the following scenarios, choose a sorting algorithm (from either selection sort, insertion sort, or merge sort) that best applies, and justify your choice. Each sort may be used more than once. If you find that multiple sorts could be appropriate for a scenario, identify their pros and cons, and choose the one that best suits the application. State and justify any assumptions you make. "Best" should be evaluated by asymptotic running time.

(a) Suppose you are given a data structure D maintaining an extrinsic order on n items, supporting two standard sequence operations: `D.get_at(i)` in worst-case $\Theta(1)$ time and `D.set_at(i,x)` in worst-case $\Theta(n \lg n)$ time. Choose an algorithm to best sort the items in D *in-place*.

(b) Suppose you have a static array A containing pointers to n comparable objects, pairs of which take $\Theta(\lg n)$ time to compare. Choose an algorithm to best sort the pointers in A so that the pointed-to objects appear in non-decreasing order.

(c) Suppose you have a *sorted array* A containing n integers, each of which fits into a single machine word. Now suppose someone performs some $\log \log n$ swaps between pairs of adjacent items in A so that A is no longer sorted. Choose an algorithm to best re-sort the integers in A.

Problem 6. Let $T[i, \ldots, j]$ be an array with $n = j - i + 1$ elements. Consider the following sorting algorithm knwon as *quim's wild sorting algorithm*:

(a) If $n \leq 2$, sort the array directly.

(b) If $n \geq 3$, "divide" the array into three intervals T[i,. . . ,k-1], T[k,. . . ,l] and T[l+1,. . . ,j], where $k = i + \lfloor n/3 \rfloor$ and $l = j - \lfloor n/3 \rfloor$.

The algorithm recursively calls itsel to first sort T[i,. . . ,l], then sort $T[k, \ldots, j]$, and finally sort $T[i, \ldots, l]$ again.

Prove that this algorithm is correct. Give a recurrence for its running time and solve it.

Problem 7. Write an algorithm with time complexity $O(\lg n)$ that, given two sorted arrays with n elements each, and an integer k, find the k^{th} smallest element of all $2n$ elements.

Problem 8. We are given a vector of vector of vectors $V = [0 \ldots n-1][2]$ containing information of n individuals. Each position $V[i]$ stores the two surnames of the i^{th} individual (this is the Iberian peninsula): $V[i][0]$ stores the first surname, $V[i][0]$ stores the second surname.

We want to sort the vector in the usual order (in the Iberian peninsula anyway): The individuals with smallest first surname first. In case of a draw, the individuals with smaller second surname go first. Assume that no two individuals share the same pair of surnames.

For example, if the inital contents of V were:

	0	1	2	3
0	Garcia	Roig	Garcia	Grau
1	Pi	Negre	Cases	Negre

the final result would have to be:

	0	1	2	3
0	Garcia	Garcia	Grau	Roig
1	Cases	Pi	Negre	Negre

Among the following combinations, one and only one solves the problem in all cases. State which one and what the cose is in terms of running time and memory space:

(a) Fist we sort V with quick sort using the first surname, then we sort it with merge sort using the second surname.

(b) Fist we sort V with merge sort using the first surname, then we sort it with quick sort using the second surname.

(c) Fist we sort V with quick sort using the second surname, then we sort it with merge sort using the first surname.

(d) Fist we sort V with merge sort using the second surname, then we sort it with quick sort using the first surname.

Problem 9. Consider the following scheme for sorting a list of n-bit (the numbers have n digits when written in binary, though some leading digits may be zero).

- We first insert the item into a hashtable with chaining, with the hash function being the first k bits of the number (this is not a good hash function in general, but serves a purpose here).
- Next, we sort the individual lists of items in each slot in the hashtable with insertion sort.
- Finally, we just concatenate all of these sorted lists together.

(a) Why does this algorithm sort correctly?

(b) What is the worst-case running time of this algorithm and with what kinds of inputs is it realized?

(c) Assuming the numbers are completely random, what running time would you expect in practice? Explain?

References

Atserias, Albert et al. (2022). *Data Structure and Algorithms Problem Set*. Universitat Politécnica de Catalunya.