The University of Michigan
Electrical Engineering & Computer Science
EECS 281: Data Structures and Algorithms
Fall 2024

Lab 5: Sorting Algorithms

15/15 ✓

Instructions:
Work on this document with your group, then enter the answers on the canvas quiz.

Note:

On the Canvas quiz, you will have **three** attempts and the best score will be kept. For additional reading, read Chapters 2, 6-8 on the CLRS Introduction to Algorithms textbook. You may also find the following site helpful: datastructures.maximal.io/sorting

You MUST include the following assignment identifier in a comment at the top of every file you submit to the autograder. This includes all source files, header files, and your Makefile (if there is one). (Since there is no autograder assignment for this lab, you may ignore this.)

Project Identifier: CD7E23DEB13C1B8840F765F9016C084FD5D3F130

3 Sorting Algorithms

3. Given the array [13, 1, 3, 2, 8, 21, 5, 1], suppose we choose the pivot to be 1, the second element in the array. Which of the following could be valid partitions of the array (assume the final goal is least-to-greatest)? Make no additional assumptions about the specific partition algorithm. Select all that apply.
   - A. 1 [1, 2, 3, 5, 8, 13, 21]
   - B. 1 [13, 1, 3, 2, 8, 21, 5]
   - C. 1 [13, 3, 2, 8, 21, 5, 1]
   - D. [1] 1 [13, 3, 2, 8, 21, 5]
   - E. [13] 1 [3, 2, 8, 21, 5, 1]
   - F. [13, 1, 3, 2, 8, 21, 5] 1

4. Suppose you wanted to create a new quicksort-inspired algorithm that has worst-case Θ(n log n) complexity. Which of the following designs would be guaranteed to achieve this goal? Select all that apply.
   - A. Before partitioning, check if the input is nearly sorted, then perform insertion sort instead. X still $O(n^2)$
   - B. Only perform at most $log_2(n)$ levels of recursion, and then switch to mergesort.
   - C. Always use the middle element in the list as the pivot instead of the last. X still $O(n^2)$ with bad luck
   - D. Shuffle the elements in the array after each stage of partitioning. X still $O(n^2)$
   - E. None of the above designs would work.

5. Which of the following is/are always TRUE about sorting functions? Select all that apply.
   - A. The most optimal partitioning policy for quicksort on an array we know nothing about would be to select a random element in the array as our pivot.
   - B. The fastest possible comparison sort has a worst-case no better than O(n log n).
   - C. Heapsort is usually best when you need a stable sort. X heapsort not stable
   - D. Sorting an already sorted array of size n with quicksort takes Θ(n log n) time. X still $O(n^2)$
   - E. When sorting elements that are expensive to copy, it is generally best to use mergesort. X mergesort : additional memory
   - F. Quicksort will always sort an array faster than an elementary sort. X

6. Suppose you have a variation of insertion sort that used binary search to find the correct slot for the i^{th} number among the i - 1 numbers that have been processed so far. What is the worst-case complexity of this new insertion sort?
   - A. $\Theta(n)$
   - B. $\Theta(nlog(n))$
   - C. $\Theta(n^2)$
   - D. $\Theta(n^2log(n))$
   - E. none of the above

specially : if take first/last element as pivot ⇒ $\Theta(n^2)$
if take middle ⇒ $\Theta(n\log n)$

7. What is the best possible worst-case time complexity of sorting a singly-linked list, and what sorting algorithm is this implementation most similar to?
   - A. $\Theta(n)$, bubble sort
   - B. $\Theta(nlog(n))$, mergesort
   - C. $\Theta(nlog(n))$, heapsort
   - D. $\Theta(n^2)$, insertion sort
   - E. none of the above

8. Given an unsorted `std::vector<int>` and a number n, what is the worst-case time complexity of finding the pair of integers whose sum is closest to n, if you are only allowed $\Theta(1)$ additional memory? For example, if you were given the vector $\{12, 3, 17, 5, 7\}$ and $n = 13$, you would return the pair $\{5, 7\}$.
   - A. $\Theta(log(n))$
   - B. $\Theta(n)$
   - C. $\Theta(nlog(n))$
   - D. $\Theta(n^2)$
   - E. $\Theta(2^n)$

   1. sort $\Theta(n \log n)$
   2 . . . . ← $O(n)$
   →

9. What is the best time complexity of merging four sorted arrays, if each of them is of size n?
   - A. $\Theta(1)$
   - B. $\Theta(log(n))$
   - C. $\Theta(n)$
   - D. $\Theta(nlog(n))$
   - E. $\Theta(n^2)$

10. Consider two vectors that are NOT currently sorted, each containing n comparable items. How long would it take to display all items (in any order) which appear in either the first or second vector, but not in both, if you are only allowed $\Theta(1)$ additional memory? Give the worst-case time complexity of the most efficient algorithm.
    - A. $\Theta(log(n))$
    - B. $\Theta(n))$
    - C. $\Theta(nlog(n))$
    - D. $\Theta(n^2)$
    - E. $\Theta(n^2log(n))$

    ie find set symmetric difference
    (two ptr) : a[i] smaller ⇒ i++, output
    b[j] smaller ⇒ j++, output
    a[i]=b[j] ⇒ i,j ++

11. What is the worst-case time complexity of finding an element in a sorted `std::list` if you are limited to $\Theta(1)$ additional memory?
    - A. $\Theta(log(n))$
    - B. $\Theta(n))$
    - C. $\Theta(nlog(n))$
    - D. $\Theta(n^2)$
    - E. $\Theta(2^n)$

12. What is the worst-case time complexity of finding an element in a sorted `std::list` if the restriction of $\Theta(1)$ additional memory is removed?

    A. $\Theta(log(n))$
    B. $\Theta(n))$
    C. $\Theta(nlog(n))$
    D. $\Theta(n^2)$
    E. $\Theta(2^n)$

    一样 (linked list 无random access)

13. You are given the following six numbers:

    183 280 281 370 376 482

    You are told to insert these six numbers, in any order, into a vector of integers. This vector will then be sorted in ascending order using a variation of quicksort that always chooses the last element of the vector as the pivot. How many distinct insertion orders of these six integers would cause this variation of quicksort to run in the worst-case in terms of integer comparisons? Hint: the worst-case happens when the pivot chosen is always the smallest or the largest element at every step of the quicksort algorithm. How many times will you have to choose the pivot here?

    A. 2
    B. 32
    C. 36
    D. 64
    E. 72

    2^5
    ↑
    max/min

14. Which of the following sorting algorithms could be implemented on a doubly-linked list WITHOUT making the asymptotic worst-case complexity even worse? You must perform the sorting in-place; that is, you cannot just copy the contents of the list to an array and then use the sorting algorithm normally. Select all that apply.

    A. quicksort worsts n2
    B. heapsort ✗
    C. bubble sort ✓
    D. selection sort ✓
    E. insertion sort ✓

    portition :

15. You are given the following three snapshots, each of which represents some of the first few intermediate steps in the sorting of an array of integers.

```
281, 280, 203, 485, 388, 183, 490, 370
280, 281, 203, 485, 183, 388, 370, 490
203, 280, 281, 485, 183, 370, 388, 490
```

Which of the following sorts is currently being run on the array?

A. bubble sort
B. heapsort
C. insertion sort
D. mergesort
E. quick sort
F. selection sort

16. You are given the following three snapshots, each of which represents some of the first few intermediate steps in the sorting of an array of integers.

```
477, 487, 381, 398, 280, 101, 490, 492
477, 381, 398, 280, 101, 487, 490, 492
381, 398, 280, 101, 477, 487, 490, 492
```

Which of the following sorts is currently being run on the array?

A. bubble sort
B. heapsort
C. insertion sort
D. mergesort
E. quick sort
F. selection sort

17. You are given the following three snapshots, each of which represents some of the first few intermediate steps in the sorting of an array of integers.

```
281, 280, 203, 381, 482, 183, 445, 376
482, 381, 445, 376, 280, 183, 203, 281
445, 381, 281, 376, 280, 183, 203, 482
```

heapified

(fixed down)

Which of the following sorts is currently being run on the array?

A. bubble sort
B. heapsort
C. insertion sort
D. mergesort
E. quick sort
F. selection sort

18. Consider the following implementation of `partition` from the quicksort algorithm that was covered in lecture. Note that the last element in the array is chosen as the pivot.

```
int partition(int a[], int left, int right) {
   int pivot = --right;
   while (true) {
       while (a[left] < a[pivot])
           ++left;
       while (left < right && a[right - 1] >= a[pivot])
           --right;
       if (left >= right)
           break;
       swap(a[left], a[right - 1]);
   }
   swap(a[left], a[pivot]);
   return left;
}
```

Suppose that you had the following unsorted array:

`int[] arr = 88, 34, 77, 20, 53, 45, 12, 76, 29, 61;`

What are the contents of this array after **one call** to `partition(arr, 0, 10)`?

A. {`12, 20, 29, 34, 45, 53, 61, 76, 77, 88`}

B. {`12, 20, 29, 34, 53, 45, 88, 76, 77, 61`}

C. {`29, 34, 12, 20, 45, 53, 61, 76, 77, 88`}

D. {`29, 34, 12, 20, 53, 45, 61, 76, 88, 77`}

E. none of the above

19. Given the following elementary sorts, which would be significantly more efficient in sorting an array of data that is mostly sorted, where each element is close to its final position?

(I) bubble sort

(II) selection sort

(III) insertion sort

A. I only

B. I and II only

C. I and III only

D. II and III only

E. I, II, and III

