ShanghaiTech University

CS101 Algorithms and Data Structures
Fall 2024

Homework 2

Due date: October 16, 2024, at 23:59

1. Please write your solutions in English.
2. Submit your solutions to Gradescope.
3. Set your FULL name to your Chinese name and your STUDENT ID correctly in Gradescope account settings.
4. If you want to submit a handwritten version, scan it clearly. CamScanner is recommended.
5. When submitting, match your solutions to the problems correctly.
6. No late submission will be accepted.
7. Violations to any of the above may result in zero points.

1. (12 points) Multiple Choices

Each question has **one or more** correct answer(s). Select all the correct answer(s). For each question, you will get 0 points if you select one or more wrong answers, but you will get half points if you select a non-empty subset of the correct answers.

Write your answers in the following table.

(a)	(b)	(c)	(d)	(e)	(f)

(a) (2') Consider a table of capacity 11 using open addressing with hash function k mod 11 and linear probing. After inserting 7 values into an empty hash table, the table is below. Which of the following choices give a possible order of the key insertion sequence?

Index	0	1	2	3	4	5	6	7	8	9	10
Keys			24	47	26	3	61	15	49		

A. 26, 61, 47, 3, 15, 24, 49

B. 26, 24, 3, 15, 61, 47, 49

C. 24, 61, 26, 47, 3, 15, 49

D. 26, 61, 15, 49, 24, 47, 3

(b) (2') Consider a table of capacity 8 using open addressing with hash function $h(k) = k \bmod 8$ and probing function $H(k) = h(k) + c_1 i + c_2 i^2 \mod 8$ with $c_1 = c_2 = 0.5$. After inserting 7 values into an empty hash table, the table is below. Which of the following choices give a possible order of the key insertion sequence?

Index	0	1	2	3	4	5	6	7
Keys	14	6	10	9	25		22	15

A. 15, 22, 10, 6, 25, 14, 9

B. 10, 22, 15, 6, 14, 9, 25

C. 6, 25, 15, 14, 22, 9, 10

D. 10, 22, 15, 6, 14, 9, 25

(c) (2') Which of the following statements about the hash table are true?

A. We have a hash table of size $2n$ with a uniformly distributed hash function. If we store n elements into the hash table, then with a very high probability, there will be **no** hash collision.

B. In a hash table with a uniformly distributed hash function where collisions are resolved by chaining, an unsuccessful search (i.e. the required element does not exist in the table) takes $\Theta(1)$ on average if the load factor of the hash table is $O(1)$.

C. Lazy erasing means marking the entry/bin as erased rather than deleting it.

D. Rehashing is a technique used to resolve hash collisions.

(d) (2') Applying insertion sort and the most basic bubble sort without a flag respectively on the same array, for both algorithms, which of the following statements is/are true? (simply assume we are using swapping for insertion sort)

A. There are two for-loops, which are nested within each other.

B. They need the same amount of element comparisons.

C. They need the same amount of swaps.

D. None of the above.

(e) (2') In the lecture we have learned that different sorting algorithms are suitable for different scenarios. Which of the following statements is/are suitable for insertion sort?

A. Each element of the array is close to its final sorted position.

B. An array where only a few elements are not in its final sorted position.

C. A big sorted array with a small sorted array concatenated to it.

D. None of the above.

(f) (2') The time complexity for both insertion sort and bubble sort will be the same if: (assume bubble sort is flagged bubble sort)

A. the input array is reversely sorted.

B. the input array is a list containing n copies of the same number.

C. the input array is already sorted.

D. None of the above.

2. (11 points) Hash Table Insertions and Deletions

Consider an empty hash table of capacity 7 and with a hash function $h(k) = (3k + 6) \mod 7$. Collisions are resolved by quadratic probing with the probing function $H_i(k) = (h(k) + i^2) \mod 7$, paired with lazy erasing. We will give three kinds of instructions, which are among the set {Insert, Delete, Search}. For `Insert/Delete` instructions, you need to fill the hash table after each instruction. For `Search` instructions, write down the probing sequence (index). Use 'D' to indicate that the bin has been marked as deleted.

(a) (1') Insert 9

Index	0	1	2	3	4	5	6
Key Value						9	

(b) (1') Insert 17

Index	0	1	2	3	4	5	6
Key Value		17				9	

(c) (1') Insert 32

Index	0	1	2	3	4	5	6
Key Value		17			32	9	

(d) (1') Insert 24

Index	0	1	2	3	4	5	6
Key Value		17	24		32	9	

(e) (1') Insert 18

Index	0	1	2	3	4	5	6
Key Value		17	24		32	9	18

(f) (1') Search 18

Solution: 4, 5, 1, 6

(g) (1') Delete 32

Index	0	1	2	3	4	5	6
Key Value		17	24		D	9	18

(h) (1') Insert 25

Index	0	1	2	3	4	5	6
Key Value		17	24		25	9	18

(i) (3') Suppose that the collisions are resolved by linear probing.

i. Write down the content of the hash table after Insert 9, 17, 32, 24, 18.

Index	0	1	2	3	4	5	6
Key Value		17	24		32	9	18

ii. What is the load factor λ?

Solution:

$$\lambda = \frac{5}{7}.$$

3. (6 points) Sorting practice

(a) (2') Run Insertion Sort for array $\{1, 4, 8, 3, 5, 2, 7, 6\}$. Write down the array **after each** outer iteration.

```
for(int k = 1; k < n; k++){
    for(int j = k; j > 0; j--){
        if( array[j- 1] > array[j] )
            swap(array[j- 1], array[j]);
        else
            break;
        }
    print(array);
}
```

Solution:

1, 4, 8, 3, 5, 2, 7, 6
1, 4, 8, 3, 5, 2, 7, 6
1, 3, 4, 8, 5, 2, 7, 6
1, 3, 4, 5, 8, 2, 7, 6
1, 2, 3, 4, 5, 8, 7, 6
1, 2, 3, 4, 5, 7, 8, 6
1, 2, 3, 4, 5, 6, 7, 8

(b) (2') Run flagged bubble Sort for the array in (a). Write down the array **after each** outer iteration.

```
for(int i = n-1; i > 0; i--){
    int max_t = array[0];
    bool sorted = true;
    for(int j = 1; j <= i; j++){
        if(array[j] < max_t){
            array[j-1] = array[j];
            sorted = false;
        }
        else{
            array[j-1] = max_t;
            max_t = array[j];
        }
    }
    array[i] = max_t;
    print(array);
    if (sorted)
```

```
                break;
        }
```

Solution:
1, 4, 3, 5, 2, 7, 6, 8
1, 3, 4, 2, 5, 6, 7, 8
1, 3, 2, 4, 5, 6, 7, 8
1, 2, 3, 4, 5, 6, 7, 8
1, 2, 3, 4, 5, 6, 7, 8

(c) (2') Run Merge Sort for this array. Write down the array **after** each merge and underline the sub-array being merged.

Solution: 1,4,3,8,2,5,6,7
1,3,4,8,2,5,6,7
1,2,3,4,5,6,7,8

4. (4 points) Insertion Sort using Linked List

In the lecture, we have learnt the insertion sort implementation using array. In this question, you are required to implement insertion sort using single linked list. Since it is not easy to traverse single linked list from back to front, we can traverse from front to back instead if it is needed.

Fill in the blanks to complete the algorithm. Please note that there is at most one statement (ended with ;) in each blank line.

```
struct ListNode
{
    int val;
    ListNode *next;
    ListNode(int x) : val(x), next(nullptr) {}
};

ListNode* insertionSort(ListNode* head) {
    //if linked list is empty or only contains 1 element, return directly
    if(!head || !(head->next)){return head;}
    //create dummy node
    ListNode *dummy = new ListNode(-1);
    dummy->next = head;
    //split linked list into sorted list and unsorted list
    ListNode *tail = head;//tail of sorted list
    ListNode *sort = head->next;//head of unsorted list
    //insertion sort
    while(sort)
    {
       if(sort->val < tail->val)
       {
           ListNode *ptr = dummy;
           while(ptr->next->val < sort->val) ptr = ptr->next;
           // Your code HERE!
           // line1
           // line2
           // line3
           // line4
       }
       else
       {        //no need to insert
           tail = tail->next;
           sort = sort->next;
```

```
        }
    }
    ListNode *ans = dummy->next;
    delete dummy; dummy = nullptr;
    return ans;
}
```

Solution:

```
tail->next = sort->next; //line1
sort->next = ptr->next; //line2
ptr->next = sort; //line3
sort = tail->next; //line4
```

5. (15 points) Sorted sort.

For each of the following scenarios, choose a sorting algorithm (from either bubble 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.

Your justification will be worth more points than your choice.

(a) (5') 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 $\Theta(n \log n)$ time. Choose an algorithm to sort the items in `D` in place best.

Solution: Merge sort is not an in-place sorting algorithm, so it will not be our solution. Now consider insertion sort and bubble sort(with no improvements): both algorithms perform $\Theta(d)$ swaps, which in the given situation costs $\Theta(d \cdot n \log n)$ time in the worst case. Besides, insertion sort will perform at least n and at most $n + d$ comparisons, which takes $\Theta(n)$ time and $\Theta(n + d)$ time in the best and worst case, respectively. But bubble sort will perform $\Theta(n^2)$ comparisons, which takes $\Theta(n^2)$ time. So the time complexity of insertion sort and bubble sort based on the given data structure `D` takes $\Theta(d \cdot n \log n + n + d)$ time and $\Theta(d \cdot n \log n + n^2)$ time respectively. So insertion sort is no worse than bubble sort, choose insertion sort.

(b) (5') Suppose you have a static array A containing pointers to n comparable objects, pairs of which take $\Theta(\log n)$ time to compare. Choose an algorithm to sort the pointers in A so that the pointed-to objects appear in non-decreasing order with minimum time cost.

Solution: For this problem, reads and writes take constant time, but comparisons are expensive, $O(\log n)$. Selection and insertion sorts both perform $O(n^2)$ comparisons in the worst case, while merge sort only performs $O(n \log n)$ comparisons, so we choose merge sort.

(c) (5') Suppose you have a sorted array A containing n integers. 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.

Solution: The performance of selection sort and merge sort do not depend on the input; they will run in $\Theta(n^2)$ and $\Theta(n \log n)$ time, regardless of the input. Insertion sort, on the other hand, can break early on the inner loop, so can run in $O(n)$ time on some inputs. To prove that insertion sort runs in $O(n)$ time for the provided inputs, observe that performing a single swap between adjacent items can change the number

of inversions1 in the array by at most one. Alternatively, every time the insertion sort swaps two items in the inner loop, it fixes an inversion. Thus, if an array is k adjacent swaps from sorted, insertion sort will run in $O(n + k)$ time. For this problem, since $k = \log \log n = O(n)$, insertion sort runs in $O(n)$ time, so we choose insertion sort.