Demonstration of Algorithm Midterm Exam

2022/05/25

1.

(a) (5%) Choose the best sorting method from the set of quicksort, insertion sort, selection sort and mergesort, if the input is given incremental. Briefly explain your answer.

Time complexities of sorting an incremental input on each method

Method	Performence
quicksort	$O(n^2)$
insertion sort	$O(n)$
selection sort	$O(n^2)$
mergesort	$O(n \lg n)$

1.

(b) (5%) To sort a set of numbers, name a strategy to provide a stable result if there is only quicksort available.

We can sort the numbers by adding the index information to it.

1.

(c) (5%) Is $\Theta(f(n)) + O(f(n)) = \Theta(f(n))$ correct? Explain your answer.

Your Proof need to get the two equations below:

(1) $\mathbf{\Theta(f(n)) + O(f(n)) = \Omega(f(n))}$

(2) $\mathbf{\Theta(f(n)) + O(f(n)) = O(f(n))}$

or just prove it via the definitions of Θ and big-O

1.

(d) (5%) To store the customer data for a bank, what could be the best data structure? Different from the typical case, let us assume that the data will not be changed very often. That is, there is no much insertions and deletions.

We can define a good hash function to have as little collisions as possible.

1.

(e) (5%) Use your own words to explain the following piece of proof,

$$n! \leq \ell \leq 2^h$$

$$h \geq \lg(n!) = \Omega(n \lg n)$$

which is used to prove the lower bound of the worst case for comparison sort.

You need to explain the reason why we get the inequality $\mathbf{n! \leq \ell \leq 2^h}$, and how to get the second equation $\mathbf{h \geq \lg(n!) = \Omega(nlgn)}$

1.

(f) (10%) On double hashing, compare two approaches: (1) the first hash function is chosen to be a multiplication method and the second hash function remains to be a division method and (2) the first hash function remains to be a division method and the second hash function is chosen to be a multiplication method. Which one makes sense more and why?

2.

(a) (5%) $T(n) = \sqrt{5}T(n/5) + n + n^2$

(b) (5%) $T(n) = 2T(n/3 - 5) + n^2/\lg n$

2.

(c) (10%) Suppose we apply the Master theorem to of the following two recurrence equations

$$T(n) = aT\left(\frac{n}{b}\right) + n^k \text{ and } T(n) = aT\left(\frac{n}{b}\right) + n^{2k}$$

and obtain the same result. What conclusion you can draw over here?

All the conclution we may have:

(1) If $2k < \log_b a$, both fall into case 1 of M.T.

(2) If $k > \log_b a$, both fall into case 3 of M.T.

(3) $a = 1,\ k = 0$, both fall into case 2 of M.T.

(4) $k < \log_b a < 2k$, i.e. apply case 1 for the left and case 3 for the right

3.

(15%) We have the following code for the quicksort algorithm implementation which combines insertion sort and quicksort. (Note that this implementation is different from the version we discussed in class and it does not use the median-3 strategy to find pivots.)

```
ISS2-QUICKSORT(A, p, r, M)
1  if   r − p + 1 ≥ M
2      then  q ← PARTITION(A, p, r)
3            ISS2-QUICKSORT(A, p, q − 1)
4            ISS2-QUICKSORT(A, q + 1 ,r)
5  else if  p < r
6      then  INSERTION-SORT(A, p, r)
```

3.

(a) (10%) If someone chooses a big M such as $M = n/4$ where n is the number of inputs, what could be the time complexity for this quicksort implementation? Discuss both the best and worst cases.

Overall, we have $T(n) = 4T\left(\frac{n}{4}\right) + 2n = O(n)$ for best case in insertion sort. On average and worst case we have, $T(n) = O(n^2)$ for insertion sort.

3.

(b) (5%) If someone chooses a small M such as $M = 2$, what could the time complexity again?

The time complexity would be very close to the original quicksort algorithm

4.

We have the following codes for heap manipulation. Answer the questions after that.

```
MAX-HEAPIFY(A, i)
 1  l → LEFT(i)
 2  r → RIGHT(i)
 3  if   l ≤ heap-size[A] and A[l] > A[i]
 4  then largest ← l
 5  else largest ← i
 6  if   r ≤ heap-size[A] and A[r] > A[largest]
 7  then largest ← r
 8  if   largest ≠ i
 9     then SWAP(A[i], A[largest])
10          MAX-HEAPIFY(A, largest)
```

```
BUIDE-MAX-HEAP(A)
1  heap-size[A] ← length[A]
2  for i ← ⌊length[A]/2⌋ downto 1
3      do MAX-HEAPIFY(A, i)
```

```
HEAP-INCREASE-KEY(A, i, key)
1  if key < A[i]
2      then error "new key is smaller than current key"
3  A[i] ← key
4  while i > 1 and A[Parent(i)] < A[i]
5      do SWAP(A[i], A[Parent(i)])
6          i ← Parent(i)
```

4.

(a) (5%) If an array $(a_1, a_2, \dots, a_n)$ is a max-heap, now we add one more element to make the array to be $(a_1, a_2, \dots, a_{n+1})$ how can we ensure the new array is also a max-heap?

The new element a_{n+1} should follow the max-heap $a_{n+1} \leq a_{\lfloor \frac{n+1}{2} \rfloor}$
Else, call the function, HEAP-INCREASE-KEY(A, heap-size[A], a_{n+1})

(b) (5%) What is the time complexity of (a)?

We insert the value at the end of the tree, and traverse up to remove the violated property $O(\lg n)$ of max-heap,

5.

(10%) In the counting sort implementation, if the code in line 4 is wrongly put by

for $i \leftarrow 1$ **to** length[A]

What will happen?

The outcome will be same, and looks sorted, but does not have stable property anymore

```
COUNTING-SORT(A, B)
1   for j ← 0 to m      do C[j] ← 0
2   for i ← 1 to length[A]      INCR(C[A[i]])
3   for j ← 1 to m      do C[j] = C[j-1]+C[j]
4   for i ← length[A] downto 1
5                 do B[C[A[i]]] ← A[i]
6               DESC(C[A[i]])
```

6.

(a) (8%) Demonstrate what happens when we insert the keys 10, 9, 8, 1, 2, 3, 15, 14, 13, 12 into a has h table of size 11 with collision resolved by both linear probing and double hashing The hash functions are defined by $h_1(k) = k \bmod 11$ and $h_2(k) = 1 + (k \bmod 7)$

Linear probing : $h(k,i) = (h(k) + i) \ mod \ m$

Double probing : $h(k,i) = (h_1(k) + ih_2(k)) \ mod \ m$

6.

(b) (5%) How many probes can we save on average, if we shift from linear probing to double hashing

Step1 : Find the total probe of Linear Hashing

Step2 : Find the total probe of Double Hashing

Step3 : Calculate on average

6.

(c) (7%) In double hashing, what may go wrong if both hash functions share the same divisor such as

$$h(k,i) = (h_1(k) + ih_2(k)) \ mod \ m$$

- **When $h_2(k)$ value is 11, it will be a loop.**