DSA Midterm Exam 2020

答案仅供参考!

Q1

Which of the following is (are) correct?

- [x] n^2 in Theta (n^2)
- [ ] n^3 in $O(n^2)$
- [x] n^2 in $O(n^3)$
- [ ] n^3 in Theta (n^2)
- [ ] n^2 in Theta (n^3)
- [x] n^2 in $O(n^2)$

Q2

What is the minimal number of leaves that a decision tree for some sorting algorithm for an input of size n has?

- ○ $2^{(n-1)}$
- ◉ $n!$
- ○ $n\log(n)$
- ○ 2^n

Q3

What is the result of adding the key 10 to the max-heap [7,6,5,4,3,2,1,0,0,0] with initially heap size 7?

We apply the adding-operation `on the fly`.

(But see attached pseudo-code)

Algorithm heapIncreaseKey(H, i, k):
    if $k < H[i]$ **then**
        return error
    $H[i] := k$
    while $i > 1$ **and** $H[\text{parent}(i)] < H[i]$ **do**
        swap$(H[\text{parent}(i)], H[i])$
        $i :=$ parent(i)

Algorithm heapInsert(H, k):
    $H.\textit{heap-size} := H.\textit{heap-size} + 1$
    $H[H.\textit{heap-size}] := -\infty$
    HeapIncreaseKey$(H, H.\textit{heap-size}, k)$

- ○ [10,7,6,5,2,3,4,1,0,0] with heapsize 10.
- ◉ [10,7,5,6,3,2,1,4,0,0] with heapsize 8.
- ○ [7,5,6,1,2,3,4,10,0,0] with heapsize 8.
- ○ [10,7,6,5,4,3,2,1,0,0] with heapsize 8.

Q4

In heapsort we use the subroutine MaxHeapify(see the attached pseducode, which takes as input an array A and an index i in A. What is the worst-case time complexity of MaxHeapify (One or more options are correct)

$$
\begin{array}{l}
\textbf{Algorithm } \text{MaxHeapify}(A, i): \\
\quad l := \text{left}(i) \\
\quad r := \text{right}(i) \\
\quad \textbf{if } l \leq A.\textit{heap-size} \textbf{ and } A[l] > A[i] \textbf{ then} \\
\quad\quad \textit{largest} := l \\
\quad \textbf{else} \\
\quad\quad \textit{largest} := i \\
\quad \textbf{if } r \leq A.\textit{heap-size} \textbf{ and } A[r] > A[\textit{largest}] \textbf{ then} \\
\quad\quad \textit{largest} := r \\
\quad \textbf{if } \textit{largest} \neq i \textbf{ then} \\
\quad\quad \text{swap}(A[i], A[\textit{largest}]) \\
\quad\quad \text{MaxHeapify}(A, \textit{largest})
\end{array}
$$

- [x] O (h) with h the height of the max-heap
- [ ] O (n) with n the number of keys of the max-heap
- [ ] O (log(h)) with h the height of the max-heap
- [x] O (log(n)) with n the number of keys of the max-heap
- [ ] O (n/2) with n the number of keys of the max-heap

Q5

> We consider two statements.
> Statement A is: $2^{(n+1)}$ is in $O(2^n)$.
> Statement B is: 2^{2n} is in $O(2^n)$.
> Which of the following holds?

- ○ B is true but A is false.
- ○ A and B are both true.
- ◉ A is true but B is false.
- ○ A and B are both false.

Q6

Which of the following recurrence equations(one or more) describe(s) correctly a function T for the worst-case time complexity of merge sort (assuming that T(1)=1)?

- [x] 2T(n/2) + 4n
- [ ] T(n/2) + n
- [ ] 4T(n/2) + 2n
- [x] 2T(n'2) + n
- [x] 2T(n/2) + cn with c a constant

Q7

Why is quicksort a good sorting algorithm?

- (●) Because it is rather efficient, despite the fact that it does not have an optimal worst case time complexity.
- ( ) Because it is the default sorting algorithm in the libraries of major programming languages.
- ( ) Because it is a recursive algorithm.
- ( ) Because it has good worst-case time complexity.

Q8

Which one of the following arrays is(are) a max-heap?

- [x] [7,6,3,5,4,2,1]
- [x] [7,4,6,3,1,5,2]
- [ ] [7,6,4,1,2,5,3]
- [ ] [7,6,2,5,4,3,1]
- [ ] [1,2,3,4,5,6,7]

Q9

We consider the algorithm partition used in quicksort (see the attached pseudo-code). What is the worst-case time complexity?

$$
\begin{array}{l}
\textbf{Algorithm } \mathsf{partition}(A, p, r): \\
\quad x := A[r] \\
\quad i := p - 1 \\
\quad \textbf{for } j = p \textbf{ to } r - 1 \textbf{ do} \\
\quad\quad \textbf{if } A[j] \leq x \textbf{ then} \\
\quad\quad\quad i := i + 1 \\
\quad\quad\quad \text{exchange } A[i] \text{ with } A[j] \\
\quad \text{exchange } A[i+1] \text{ with } A[r] \\
\quad \textbf{return } i + 1
\end{array}
$$

- ○ The worst-case time complexity is in Theta (n^2) with n the number of elements in the array A.
- ○ The worst-case time complexity is in Theta (n log(n)) with n the number of elements in the array A.
- ◉ The worst-case time complexity is in Theta (n) with n the number of elements in the array A.
- ○ The worst-case time complexity is in Theta (log (n)) with n the number of elements in the array A.

Q10

We consider the algorithm MaxHeapify(see attached pseudo-code) and we consider an execution of MaxHeapify with input an array A of length n, and an index i which is at least n/2.

What is the time complexity of such an execution?

Algorithm MaxHeapify(A, i):

$l :=$ left(i)

$r :=$ right(i)

if $l \leq A.heap\text{-}size$ **and** $A[l] > A[i]$ **then**

$largest := l$

else

$largest := i$

if $r \leq A.heap\text{-}size$ **and** $A[r] > A[largest]$ **then**

$largest := r$

if $largest \neq i$ **then**

swap$(A[i], A[largest])$

MaxHeapify$(A, largest)$

- ○ Approximately log(n) time, with n the length of the array A.
- ◉ Elementary time.
- ○ Approximately n time, with n the length of the array A.
- ○ Approximately n/2 time with n the length of the array A.

Q11

Merge-sort is a divide-and-conquer algorithm.

Suppose we adapt merge sort by splitting the sequence into 4 more or less equal-size parts. Do we get an essentially better worst-case time complexity?

- ◉ No, because the height of the recursion tree is then still proportional to log(n).
- ○ Yes, we get an algorithm which is twice as fast because the recursion tree has smaller height.
- ○ No, because then the merge-procedure takes more time,
- ○ Yes, because the merge-procedure takes smaller inputs.

Q12

We consider bucket sort (see the attached pseudo-code description).

What is the worst-case running time of bucket sort?

Algorithm bucketSort(A):
  $n := A.length$
  new array $B[0 \ldots n-1]$
  for $i := 0$ **to** $n-1$ **do**
    make $B[i]$ an empty list
  for $i := 1$ **to** n **do**
    insert $A[i]$ into list $B[\lfloor n \cdot A[i] \rfloor]$
  for $i := 0$ **to** $n-1$ **do**
    insertionSort($B[i]$)
  concatenate $B[0], B[1], \ldots, B[n-1]$

- ○ It is in Theta (n^2), because worst-case sorting is always in n^2.
- ◉ It is in Theta (n^2), because we may call insertion sort, which is quadratic, on an input of size n, namely if (floor of nA[i]) is the same for all indices i.
- ○ It is in Theta (n^3), because we may call insertion sort which is quadratic n times.
- ○ It is in Theta (n), because the for-loops are not nested.

Q13

An algorithm with worst-case time complexity Theta(nlog(n)) is always, for any input, faster than an algorthm with worst-case time complexity in Theta(n^2).

Is this statement true of false, and why?

- ◉ False, because for a small input or for a special input an algorithm with worst-care time complexity in Theta (n^2) may perform better than an algorithm with worst-case time complexity in Theta (nlog(n)).
- ○ True, because Theta gives a strict bound.
- ○ True, because nlog(n) has a lower rate of growth than n^2.
- ○ False, because n log(n) and n^2 have the same rate of growth anyway.

Q14

We consider the algorithm for partition (see pseudo-code attached).

Apply partition to the input array A= [3,7,6,1,8,5,2,4] and indices p = 1 and r = 8.

What is A after having executed the iterations for j = 1,2,3,4 ?

Algorithm partition(A, p, r):
$x := A[r]$
$i := p - 1$
for $j = p$ **to** $r - 1$ **do**
  if $A[j] \leq x$ **then**
    $i := i + 1$
    exchange $A[i]$ with $A[j]$
exchange $A[i + 1]$ with $A[r]$
return $i + 1$

- (x) A = [3,1,6,7,8,5,2,4]
- ( ) A = [3,1,7,6,8,5,2,4]
- ( ) A = [3,7,6,4,8,5,2,1]
- ( ) A = [1,3,6,7,8,5,2,4]

Q15

We consider counting sort(see the attached pseudo-code). What happens if we change the last for-loop by letting the index j go from 1 up to `A.length`?

Algorithm countingSort(A, B, k):
new array $C[0 \ldots k]$
for $i := 0$ **to** k **do**
  $C[i] := 0$
for $j := 1$ **to** $A.length$ **do**
  $C[A[j]] := C[A[j]] + 1$
for $i := 1$ **to** k **do**
  $C[i] := C[i] + C[i - 1]$
for $j := A.length$ **downto** 1 **do**
  $B[C[A[j]]] := A[j]$
  $C[A[j]] := C[A[j]] - 1$

- ○ The change has no effect at all.
- ○ The algorithm is no longer correct.
- ○ The algorithm sorts in reverse order.
- ◉ The algorithm still sorts correctly, but is no longer stable.

Q16

What is the best-case time complexity of insertion sort?

- ○ Theta (n^2)
- ○ Theta (log(n))
- ○ Theta (n log(n))
- ◉ Theta (n)

Q17

The heapsort algorithm consists of two parts: first build a max-heap and then iterate removing elements from it. Which of the two parts is responsible for the worst case complexity of heapsort?

- ◉ The second part, because the first part is linear.
- ○ It depends on the input.
- ○ Both parts have the worst time complexity of heapsort.
- ○ The first part, because the second part is linear.

Q18

We consider an application of buildMaxHeap (see attached) to A= [1,2,3,4,5,6,7]. What is A after the first iteration of the for-loop?

Algorithm buildMaxHeap(H):

$H.heap\text{-}size := H.length$

for $i = \lfloor H.length/2 \rfloor$ **downto** 1 **do**

MaxHeapify(H, i)

- (selected) [1,2,7,4,5,6,3]
- [1,5,3,4,2,6,7]
- [3,2,7,4,5,6,1]
- [3,2,1,4,5,6,7]

Q19

Consider the following recurrence equation:

T(0) = 1 and T(n) = (n-1) + n for n > 1.

What is the big-O of T?

- T(n) in O (n)
- (selected) T(n) in O (n^2)
- T(n) in O (2^n)
- T(n) in O (n log(n))

Q20

We consider a max-heap containing n different keys. Why do we have that the height of such a max-heap is approximately log(n)?

- ○ Because a max-heap is a binary tree.
- ○ Because the keys in a max-heap are partially ordered.
- ◉ Because a max-heap is an almost-complete binary tree.
- ○ Because all elements are different.

Q21

Why does the linear time complexity of counting sort not contradict the lower bound on the worst-case complexity of comparison-based sorting?

- ○ Because the lower bound is on worst-case and the linear time complexity of counting sort is best-case.
- ○ Because the lower bound is linear time.
- ◉ Because counting sort is not comparison-based.
- ○ Because counting sort uses additional memory.

Q22

Which of the following sorting algorithm has/have a worst-case time complexity in Theta(nlog(n))?

- [ ] counting sort
- [x] heapsort
- [ ] insertion sort
- [ ] bucket sort
- [x] merge sort
- [ ] selection sort
- [ ] quicksort

Q23

We consider the algorithm for the bottom-up max-heap construction (see pseudo-code attached).

What happens if we let the loop-index increase from 1 upwards to floor of `A.lenght/2` ?

Algorithm buildMaxHeap(H):
  $H.heap\text{-}size := H.length$
  for $i = \lfloor H.length/2 \rfloor$ **downto** 1 **do**
    MaxHeapify(H, i)

○ Then we need to take the ceiling of A.length / 2 instead of the floor in order for the algorithm to be correct.

○ It works equally well,

◉ Then the algorithm is no longer correct.

○ Then we should do the recursive call of MaxHeapify on A and i-1 instead of on A and i in order for the algorithm to be correct.

Q24

Which recurrence equation correctly describes, next to T(0) =1, the worst-case time complexity of quicksort ?

○ $T(n) = 2T(n/2) + cn$

○ $T(n) = T(n) + T(0) + cn$

◉ $T(n) = T(n-1) + T(0) + cn$

○ $T(n) = 2T(n/2)$

Q25

What is the worst-case time complexity of insertion sort?

- ○ O (log n)
- ◉ Theta (n^2)
- ○ Theta (n log(n))
- ○ O (n log (n))

Q26

What is the time complexity of quicksort on an array in which all keys are identical?

- ○ Theta(n log(n)), because the array is already sorted, but you still have the recursive calls.
- ◉ Theta(n^2), because when partioning one side will always be empty.
- ○ Theta(1), because the array is trivial.
- ○ Theta(n), because the array is already sorted.

Q27

We consider insertion sort(see attached). What happens if we swap the order of the tests for the while-loop in line 5?

- ○ The algorithm then sorts in reverse order.
- ◉ We may then possibly compare A[0] with key, but A[0] does not exist.
- ○ The program then does not terminate.
- ○ It does not matter.

Q28

When is a sorting algorithm said to be stable?

- ○ If the algorithm is linear time on an array that is already sorted.
- ○ If sorting the array twice in a row does not affect the result.
- ◉ If it respects the order of equal keys in the array.
- ○ If it does not need extra memory next to the input array.

Q29

What is the running time of heapsort if the input is an array that is already sorted (in increasing order)?

- ○ The running time is then in O (log(n)) .
- ○ The running time is then in O(n).
- ○ The running time is then in O(n^2).
- ◉ The running time is then in O (nlog(n)).