Question 1 Matching (20×1 point = 20 points)

Fill in each blank represented with a number with parentheses in the sentences using the best LETEER(s) representing the corresponding term(s) from the following alternatives listed below. Or answer a **T** or **F** according to the correctness of each complete statement. Each of which could be re-used (could be the answer for more than one of **(1)~(20)**:

Alternative Answers:

A. computation	**B. data type**	**C. finite**	**D. delete**	**E. heaps**	**F. false**
G. input	**H. insert**	**I. instructions**	**J. FIFO**	**K. LIFO**	**L. linked lists**
M. resizing arrays	**N. priority queue**	**O. output**	**P. randomized queue**	**Q. queue**	
R. in-place	**S. stack**	**T. true**	**X. f(n) = Θ(g(n))**	**Y. f(n) = O(g(n))**	**Z. f(n) = Ω(g(n))**

An algorithm is a sequence of unambiguous __**(1)**__ for solving a __**(2)**__ problem, i.e., for obtaining a required __**(3)**__ for any legitimate __**(4)**__ in a __**(5)**__ amount of time.

A collection is a __**(6)**__ that stores a group of items. The main operations on a collection are insert and __**(7)**__ items. Which item to delete (remove) determines the nature of a collection. In a stack, removing the item (pop) is done in a __**(8)**__ way. In a queue, removing the item (dequeue) is in a __**(9)**__ way. In a __**(10)**__, removing the item (deleteMax or deleteMin) is done with the largest (or smallest) item.

Stacks and Queues are normally implemented by __**(11)**__ data structure(s), while priority queues are normally implemented by __**(12)**__.

A sorting algorithm is __**(13)**__ if it uses $\leqslant$ c log N extra memory.

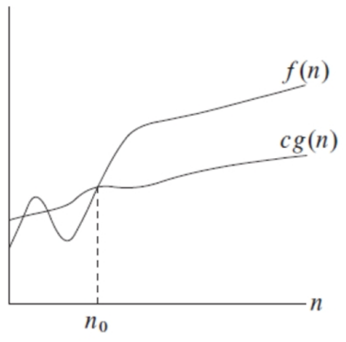
(14) If $T_1(n) = O(f(n))$ and $T_2(n) = O(f(n))$, then $T_1(n) = O(T_2(n))$.

(15) Building a heap from an array of n items requires *O* (n log n) time.

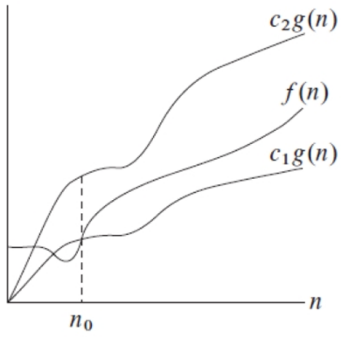
(16) For large input sizes, mergesort will always run faster than insertion sort (on the same input).

(17) To prevent too many recursive call for tiny sized array slice in mergesort or quicksort, in practice to enhance efficiency normally use cutoff to insertion sort when the length of slice is small enough.

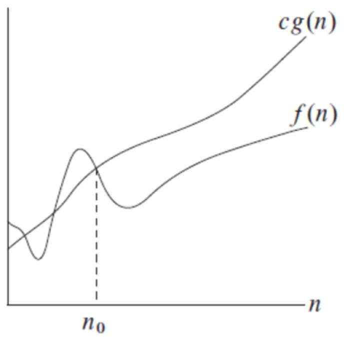
(18)~(20) are based on the following 3 plots. Please identify the corresponding asymptotic notations represented in the following 3 plots.



(18)



(19)



(20)

Question 2 On Big-O (8×1.5 points = 12 points)

For each of the functions f(N) given below, indicate the **tightest bound** possible (in other words, giving $O(2^N)$ as the answer to every question is not likely to result in many points). Unless otherwise specified, all logs are base 2. **You MUST choose your answer from the following** (not given in any particular order), each of which could be re-used (could be the answer for more than one of **(21)~(28)**:

$O(N^2)$, $O(N^{1/2})$, $O(N^{1/4})$, $O(\log^3 N)$, $O(N)$, $O(N^2 \log N)$, $O(N^5)$, $O(2^N)$, $O(N^3)$, $O(N^8)$, $O(\log^4 N)$, $O(N \log^3 N)$, $O(N^2 \log^2 N)$, $O(\log N)$, $O(1)$, $O(N^4)$, $O(N^N)$, $O(N^6)$, $O(N \log^2 N)$, $O(\log \log N)$

You do not need to explain your answer.

(21) $f(N) = N^2 \log N^2 + 2 N \log^2 N$

(22) $f(N) = (N (100N + 5 + N^3))^2$

(23) $f(N) = 1000 \log \log N + \log N$

(24) $f(N) = \log_{16}(2^N)$

(25) $f(N) = N^2 (\log N^3 – \log N)+ N^2$

(26) $f(N) = (N \log N + 2N)^2$

(27) $f(N) = N^{1/2} + \log^3 N$

(28) $f(N) = 100 \log N + N^{1/4}$

Question 3 Big-Oh and Run Time Analysis (4×3 points = 12 points)

Describe the worst case running time of the following pseudocode functions in Big-Oh notation in terms of the variable n. Showing your work is not required (although showing work may allow some partial credit in the case your answer is wrong – don't spend a lot of time showing your work.). You MUST choose your answer from the following (not given in any particular order), each of which could be re-used (could be the answer for more than one of **(29)~(32)**):
$O(n^2)$, $O(n^3 \log n)$, $O(n \log n)$, $O(n)$, $O(n^2 \log n)$, $O(n^5)$, $O(2^n)$, $O(n^3)$, $O(\log n)$, $O(1)$, $O(n^4)$, $O(n^n)$

```
(29)  void doWhat(int n, int x, int y) {
        if (x < y) {
          for (int i = 0; i < n; ++i)
```

```
            for (int j = 0; j < n * i; ++j)
               System.out.println(”y = ” + y);
      } else {
         System.out.println(”x = ” + x);
      }
   }
```

```
(30)  int doWhat(int n, int m) {
         if (n < 1) return n;
         if (n < 100) return doWhat(n - m, m);
         return doWhat(n - 1, m);
      }
```

```
(31)  void doWhat(int n) {
         int j = 0;
         while (j < n) {
            for (int i = 0; i < n; ++i) {
               System.out.println(”j = ” + j);
            }
            j = j + 5;
         }
      }
```

```
(32)  void doWhat(int n) {
         for (int i = 0; i < n * n; ++i) {
            for (int j = 0; j < n; ++j) {
               for (int k = 0; k < i; ++k)
                  System.out.println(”k = ” + k);
               for (int m = 0; m < 100; ++m)
                  System.out.println(”m = ” + m);
            }
         }
      }
```

Question 4 On Quick-sort (2×**5**points = 10 points)

Suppose a quick-sort program is in execution. The partition method has receaved a char array slice `a[low..high]` = `[J U S T F O R E X A M]`, ignore syntax symbols like single quote, please answer question **(33)** and **(34)**:

(33) *(5 points)* If choose `a[low]` as the pivot, write down the result of `a[low..high]` after the partition finished.

(34) *(5 points)* If choose `median3(a[low], a[low + (high-low)/2], a[high])` as the pivot, write down the result of `a[low..high]` after the partition finished.

Question 5 h-sorting in Shell Sort (3 * 4 points = 12 points)

An h-sorted array is h interleaved sorted subsequences. Shellsort is to h-sort an array sucecessively with a decreasing sequence of values of h. Suppose the content of `char[] a` is `[J U S T M I D T E R M E X A M]`, ignore syntax symbols like single quote, please write down the following results in a sucecessive series :

(35) *(4 points)* 13-sort of array **a**;

(36) *(4 points)* 4-sort of the result of question **(35)**;

(37) *(4 points)* 1-sort of the result of question **(36)**.

Question 6 On Binary Min Heaps (7 + 3 = 10 points)

(38) (*7 points)* Draw the binary min heap that results from inserting the integers: 7, 5, 6, 3, 8, 2 in that order into an initially empty binary min heap. You do not need to show the array representation of the heap. You are only required to show the final tree, although drawing intermediate trees may result in partial credit. If you draw intermediate trees, please circle your final result for any credit.

(39) *(3 points)* Draw the result of one deleteMin call on your heap drawn at the end of question **(38) with intermediate trees** .

Question 7 Programming (3 * 10 points = 30 points)

(40) (*10 points*) Complete the Java method so that it properly implements insertion sort. Make sure the method is complete (no any extra helping method needed) and the result is in-place (do not use another array) .

```
public static int[] insertionSort (int[] a) {
   for (int i = 1; i < a.length; i++) {
      // YOUR CODE HERE
      // …
      // …
   } // end of the for-loop
   return a;
}
```

(41) *(10 points)* Complete the Java method so that it properly implements selection sort. Make sure the method is complete (no any extra helping method needed) and the result is in-place (do not use another array). Notice there are two places to add code.

```
public static int[] void selectionSort (int[] a) {
   for (int i = 0; i < a.length; i++) {
      int idx = i;  // index_of_least
      int j;
      for (j = i+1; j < a.length; j++) {
         // YOUR CODE HERE #1
         // …
         // …
```

```
        } // end of the inner for-loop

        // YOUR CODE HERE #2
        // …
        // …
    } // end of the outer for-loop
}
```

(42) *(10 points)* Complete the helping method binarySearch so that it helps to properly implement binary search in a sorted array for the method indexOf. Both methods return the index found in the array or -1 when not found. Make sure the method binarySearch is complete (no any extra helping method needed).

```
public static <E extends Comparable<E>>
int indexOf (E x, E[] a) {
   // a is sorted by comparing with compareTo()
   return binarySearch( x, a, 0, a.length-1 );
}

private static <E extends Comparable<E>>
int binarySearch (E x, E[] a, int low, int high) {
   // Tips: use x.compareTo(y) to compare 2 Es

   // YOUR CODE HERE
   // …
   // …
}
```