Question 1. (5 points) If the merging algorithm used in mergesorting is implemented inefficiently and makes $N \log N$ comparisons rather than N, then the time complexity of the resulting mergesort would be (choose the best answer):

1. $O(N \log(\log N)))$
2. $O(N \log N)$
3. $O(N \log N \log(\log N))$
4. $O(N(\log N)^2)$
5. $O(N(\log(\log N))^2)$

Question 2. (5 points) If the merging algorithm used in mergesorting is implemented inefficiently and makes $N\sqrt{N}$ comparisons rather than N, then the time complexity of the resulting mergesort would be (choose the best answer):

1. $O(N\sqrt{N})$
2. $O(N\sqrt{N} \log N)$
3. $O(N^2)$
4. $O(N^2 \log N)$
5. $O(N^2\sqrt{N})$

Question 3. (5 points) Which one of the choices below correctly ranks the functions listed by increasing order of growth (i.e., the slowest-growing first, the fastest-growing last)?

1. $(\log \log n)^2$, $\sqrt{n}$, $(\log n)^{0.7}$, $\log(n!)$, $n^{1.1}$, n^2, 2^n, $n!$
2. $(\log n)^{0.7}$, $(\log \log n)^2$, $\sqrt{n}$, $n^{1.1}$, $\log(n!)$, n^2, 2^n, $n!$
3. $(\log n)^{0.7}$, $(\log \log n)^2$, $\sqrt{n}$, $\log(n!)$, $n^{1.1}$, n^2, 2^n, $n!$
4. $(\log \log n)^2$, $(\log n)^{0.7}$, $\sqrt{n}$, $\log(n!)$, $n^{1.1}$, n^2, 2^n, $n!$
5. $(\log \log n)^2$, $(\log n)^{0.7}$, $\sqrt{n}$, $n^{1.1}$, $\log(n!)$, n^2, 2^n, $n!$

Question 4. (5 points) This question is about the binary heap, using an array representation as covered in class. Assume an initially empty heap on which the following operations are carried out (in that order):

```
insert H, insert W, insert L, insert E, remove max, insert R, insert C,
insert Q, remove max, insert A, insert B, remove max
```

After the above sequence of operations, the array contents starting from position 1 look like (choose one):

1. L E H A C B
2. L E H C B A
3. L H E C B A
4. L H E B C A
5. L E H B C A

Question 5. (5 points) Show the array contents resulting from using the linear-time binary heap construction algorithm (covered in class) on an input array containing:

`A B C D E F G H I J K`

where we ignored position 0 (i.e., A is at position 1, B at position 2, etc).

1. K J I H G F E D C B A
2. K J G I E F C H D B A
3. K J G E I C F D H A B
4. K J G E I C F H D B A
5. K J I G H E F D B A C

Material for Questions 6 and 7. In what follows A is an array representing a binary heap that contains N distinct keys where N is a power of 2; we call the heap in A the *main heap*. A much smaller second array a is large enough to contain a binary heap of only n keys, where $n \leq \log N$; we call the heap in a the *auxiliary heap*. The auxiliary heap is initially empty. In what follows, whenever we say "insert in the auxiliary heap a key x from the main heap", we implicitly imply storing, alongside every such x, its position in the main heap (i.e., its position in array A). None of what is described below causes any modification to the main heap: It only reads information from the main heap, all the modifications described occur in the auxiliary heap.

- Insert, in the auxiliary heap, the key at the root of the main heap.
- Repeat $n - 1$ times the following:
  - Do a remove-max operation on the auxiliary heap, say it returns the key at node v of the main heap.
  - Insert, in the auxiliary heap, the keys of the two children of v in the main heap.
- Return the maximum element in the auxiliary heap.

Question 6. (5 points) The key returned by the last step of the above algorithm is (choose the best answer)

1. A key whose depth in the main heap is between $n/2$ and n
2. A key whose depth in the main heap is n
3. The kth largest key in the main heap, where k is between $n/2$ and n
4. The nth largest key in the main heap
5. A key that need not satisfy any of the above four choices

Question 7. (5 points) The time taken by the above algorithm depends on n, not on N. It is closest to being (choose the best answer)

1. $O(n)$
2. $O(n \log(\log n))$
3. $O(n \log n)$
4. $O(n(\log n)^2)$
5. $O(n^2)$

Material for Questions 8, 9, 10. Let A be an array of N (possibly negative) integers, with $A[0] = 0$. For every pair of indices i, j $(i \leq j)$, we define $S[i, j]$ as follows

$$S[i, j] = \sum_{k=i}^{j} A[k]$$

We seek an algorithm that computes and returns the largest $S[i, j]$ value in $O(N)$ time (so it obviously must avoid computing all the $S[i, j]$'s). The below algorithm does this, but some of its steps contain alternative choices that are listed between boldface brackets, like this: **{**Choice1, Choice2, Choice3**}**. For each occurrence of such choices, you need to select the choice that makes the algorithm correct.

Sketch of Algorithm Steps

1. Compute the N entries of an array L of integers such that
   $L[i] = \mathbf{\{}\ \sum_{k=0}^{i} A[k]\ ,\ \min_{k=0}^{i} A[k]\ ,\ \max_{k=0}^{i} A[k]\ \mathbf{\}}$
2. Compute the N entries of an array M of integers such that
   $M[i] = \mathbf{\{}\ \sum_{k=0}^{i} L[k]\ ,\ \min_{k=0}^{i} L[k]\ ,\ \max_{k=0}^{i} L[k]\ \mathbf{\}}$
3. Compute the N entries of an array D of integers such that $D[i] = L[i] - M[i]$
4. Compute and return integer R such that
   $R = \mathbf{\{}\ \sum_{k=0}^{N-1} D[k]\ ,\ \min_{k=0}^{N-1} D[k]\ ,\ \max_{k=0}^{N-1} D[k]\ \mathbf{\}}$

Question 8. (5 points) The correct choice for Step 1 of the algorithm is

1. $\sum_{k=0}^{i} A[k]$
2. $\min_{k=0}^{i} A[k]$
3. $\max_{k=0}^{i} A[k]$

Question 9. (5 points) The correct choice for Step 2 of the algorithm is

1. $\sum_{k=0}^{i} L[k]$
2. $\min_{k=0}^{i} L[k]$
3. $\max_{k=0}^{i} L[k]$

Question 10. (5 points) The correct choice for Step 4 of the algorithm is

1. $\sum_{k=0}^{N-1} D[k]$
2. $\min_{k=0}^{N-1} D[k]$
3. $\max_{k=0}^{N-1} D[k]$

Date due: Monday October 2, 2017 **at the beginning of class**