# University of Waterloo
# CS240 Winter 2025
# Assignment 2

**Due Date: Tuesday, February 4 at 5:00pm**

Please read `https://student.cs.uwaterloo.ca/~cs240/w25/assignments.phtml#guidelines` for guidelines on submission. **Each question must be submitted individually to Crowdmark.** Submit early and often.

**Grace period:** submissions made before 11:59PM on February 4 will be accepted without penalty. Your last submission will be graded. Please note that submissions made after 11:59PM **will not be graded** and may only be reviewed for feedback.

**Notes:**

- Logarithms are in base 2, if not mentioned otherwise.
- If you use MSD or LSD radix sort in this assignment, you are free to choose any value for radix $R$. If you choose radix $R$ other than 10 (standard decimal representation), you must account for the time it takes to transform a number in decimal representation to base $R$.

## Question 1 [2+3+5=10 marks]

**a)** Let $H$ be a max-heap of size 13 with unique elements. On which level(s) of the heap can the smallest item be located? Recall that the topmost level in the heap is 0. No explanation is needed.

**b)** Consider *NewHeapify* algorithm below, which calls *fix-down* subroutine described in lecture notes. Does this algorithm produce the max-heap order in the input array $A$? Give a proof or a counter-example.

```
NewHeapify(A)
A: an array
1.   n ← A.size()
2.   for i ← 0 to parent(last())
3.        fix-down(A, n, i)
```

**c)** Consider a heap of size $n$ implemented with an array $H$. Let the $k$th ancestor of the node at index $i$ of the array be the ancestor of $i$ that is separated from $i$ by $k$

links. For example, the parent of $i$ is the first ancestor of $i$. Design an algorithm $isAncestor(x, H, i)$ which takes as an input heap array $H$, a valid index $i$ in the array and a value $x$. This algorithm should return true if key $x$ is stored at $k$th ancestors of node $i$ for some $k \in \mathbb{Z}$, $k \geq 0$, and return false otherwise.

In order to obtain full marks, your algorithm should be as efficient as possible. You **must** write pseudo-code for this question. In addition, as for any other algorithm you provide in assignments, you must describe the algorithm, briefly justify correctness, and analyse its running time complexity in terms of $n$. Be precise about the indices your algorithm explores.

Note: For this problem, you may assume that taking a log of any number or raising 2 to any integer power takes a constant amount of time (not true in general for our idealized computer model).

## Question 2 [5 marks]

Consider the following algorithm, which takes as an input a bitstring $w$ of length $n$. Furthermore, $w$ has exactly 3 bits equal to 1. What is the average case running time of the algorithm below? Note that on line 6, if $count = 2$, then the for loop on lines 6-7 will execute $n^2$ times.

```
Mystery(w)

w is a bitstring of length n ≥ 3 containing exactly three bits equal to 1
1.  count ← 0
2.  for i = 1 to n − 1
3.      if w[i − 1] > w[i]
4.          count ← count + 1
5.  sum ← 0
6.  for i = 1 to n^count
7.      sum ← sum + 1
8.  return sum
```

## Question 3 [2+5=7 marks]

Consider the algorithm below, where *random(k)* returns an integer from the set of $\{0, 1, \ldots, k-1\}$ uniformly and at random.

```
ArrayAlg(A, n)
A: an array storing numbers
1.  if n = 1
2.      return
3.  i ← random(n) + 1
4.  for j = 1 to i do
5.      print(A[j − 1])
6.  ArrayAlg(A, n − 1)
```

**a)** Provide a tight ($\Theta$) bound for the worst-case (i.e. the worst luck) running time of *ArrayAlg* on array $A$ of size $n$. Justify.

**b)** Let $T^{exp}(n)$ be the expected running time of *ArrayAlg* for an input of size $n$. Show how to derive a recurrence relation for $T^{exp}(n)$ and then solve it. Express $T^{exp}(n)$ using big-O asymptotic notation. Your bound must be asymptotically tight, but you need not prove that it is tight.

## Question 4 [2+5=7 marks]

Consider the algorithm below, which is a variant of *QuickSort*. Here, subroutine *use-one-key-comparison* is unknown but uses exactly one key-comparison, and algorithm *partition* is the same algorithm as shown in class. Now let $T^{best}(n)$ and $T^{worst}(n)$ be the best and worst case number of key-comparisons performed by *mysteryQS* for an array of size $n$. Note that we are only counting key-comparisons performed by the sub-routine *use-one-key-comparison*.

```
mysteryQS(A, n ← A.size)
 if n > 1
      p ← choose-pivot(A)
      i ← partition(A, p)
      mysteryQS(A[0, 1, ..., i − 1])
      mysteryQS(A[i + 1, ..., n − 1])
      for j = 0 to i do
          for k = i to n − 1 do
              use-one-key-comparison()
```

**a)** Show that $T^{\text{worst}}(n) \in \Omega(n^2)$.

**b)** Show that $T^{\text{best}}(n) \in O(n^2 \log n)$.

Remark: Obviously not both (a) and (b) can be tight. To keep the assignment shorter we did *not* ask you to get the tight bounds here.

## Question 5 [5 marks]

A student designed a data structure and named it an `almost-priority-queue`. This data structure allows two operations: `insert` and `extract_almost_Max`, where `extract_almost_Max` outputs either the largest priority or the second largest priority item. Also, `extract_almost_Max` does not tell you whether it extracted the largest or second largest priority item. In case the data structure has only one element, `extract_almost_Max` extracts that element. The student claims that the worst case running time of both `insert` and `extract_almost_Max` is $o(\log n)$. Prove that the student has made a mistake in the running time analysis of their data structure.

## Question 6 [2+2+2+3=9 marks]

**a)** Show the contents of array $A = [54, 121, 38, 82]$ after one round of LSD sort, where one round means one application of single digit bucket sort. No explanation is required.

**b)** Perform MSD sort on array $A = [736, 212, 213, 376, 354, 850]$. For each number, underline the digits (if any) which are not examined by MSD sort. No explanation is required.

**c)** Provide a $\Theta$ bound on the running time of bucket-sort when sorting an array of size $n^3$ containing elements in the range $[1, \ldots, n^5]$. Briefly explain.

**d)** Explain how to sort $n$ integers in the range $[0, n^{10})$ in $O(n)$ time.