1 Heap Sort

Consider the following array $A$:

$$[4 \ 3 \ 9 \ 10 \ 14 \ 8 \ 7 \ 2 \ 1 \ 7]$$

1. Interpret $A$ as a binary tree as in the lecture (on heaps).

**Solution.**

![Binary Tree Diagram]

2. Run Create-Heap() on the initial array. Give the sequence of node exchanges. Draw the resulting heap.

**Solution.** The resulting heap looks as follows:

![Heap Diagram]

The sequence of node exchanges are: $14 \leftrightarrow 3$, $3 \leftrightarrow 7$, $4 \leftrightarrow 14$, $4 \leftrightarrow 10$  

3. What is the worst-case runtime of Heapify()?
Solution. As discussed in the lecture, Heapify() runs in time $O(\log n)$. This corresponds to the maximum height of a complete binary tree on $n$ elements.

4. Explain how heap sort uses the heap for sorting. Explain why the algorithm has a worst-case runtime of $O(n \log n)$.

Solution. See lecture. Please let us know (Drop-ins, office hours, etc.) if this is not clear.

5. Give an array of length $n$ so that heap-sort runs in $O(n)$ time on $A$.

Solution. For example, an array with $A[i] = 1$, for all $0 \leq i \leq n - 1$. Then Heapify() always runs in time $O(1)$, since no recursive calls are needed.

2 Merge Sort

Illustrate how the Mergesort algorithm sorts the following array using a recursion tree:

11 7 2 5 9 6 1

Solution.
3 Quick Sort

Consider an array \( A \) of length \( n \) so that \( A[i] = n - i \). For example, for \( n = 10 \) we are given the following array:

\[
A = 10 \ 9 \ 8 \ 7 \ 6 \ 5 \ 4 \ 3 \ 2 \ 1 .
\]

The goal is to sort \( A \) in non-decreasing order which in this case is equivalent to reversing it. The pivot plays a central role in Quicksort. Consider the following options as a choice for the pivot:

1. The right-most position.
2. The element at position \( \lceil n/2 \rceil \).
3. The left-most position.

For each of these options, what is the runtime of Quicksort on \( A \)? State your answers using \( \Theta(.) \)-notation. Justify your answers.

Solution.

1. In this case, the pivot is always the smallest element of the subarray. Every array of length \( k \) considered is then split into an array of length \( k - 1 \), the pivot, and an empty array. This yields a runtime of \( \Theta(n^2) \).

2. This is a very good split as every array of length \( k \) is split roughly two equal halves. This yields a runtime of \( \Theta(n \log n) \).

3. Similar to the first case, this leads to one empty subarray. The runtime is therefore \( \Theta(n^2) \).

✓

4 Countingsort and Radixsort

1. Illustrate how Countingsort sorts the following array:

\[
4 \ 2 \ 2 \ 0 \ 1 \ 4 \ 2
\]

Solution. See lectures.

✓

2. Illustrate how Radixsort sorts the following binary numbers:

\[
100110 \ 101010 \ 001010 \ 010111 \ 100000 \ 000101
\]

Solution.

\[
100110 \quad 100110 \quad 100000 \quad 100000 \quad 100000 \quad 100000 \quad 000101 \\
101010 \quad 101010 \quad 000101 \quad 101010 \quad 000101 \quad 000101 \quad 001010 \\
001010 \rightarrow 001010 \rightarrow 100110 \rightarrow 001010 \rightarrow 100110 \rightarrow 100110 \rightarrow 010111 \\
010111 \rightarrow 100000 \rightarrow 101010 \rightarrow 000101 \rightarrow 010111 \rightarrow 101010 \rightarrow 100000 \\
100000 \quad 010111 \quad 001010 \quad 100110 \quad 101010 \quad 001010 \quad 100110 \\
000101 \quad 000101 \quad 010111 \quad 010111 \quad 001010 \quad 010111 \quad 101010
\]
3. Radixsort sorts an array $A$ of length $n$ consisting of $d$-digit numbers where each digit is from the set $\{0, 1, \ldots, b\}$ in time $O(d(n + b))$.

We are given an array $A$ of $n$ integers where each integer is polynomially bounded, i.e., each integer is from the range $\{0, 1, \ldots, n^c\}$, for some constant $c$. Argue that Radixsort can be used to sort $A$ in time $O(n)$.

Hint: Find a suitable representation of the numbers in $\{0, 1, \ldots, n^c\}$ as $d$-digit numbers where each digit comes from a set $\{0, 1, \ldots, b\}$ so that Radixsort runs in time $O(n)$. How do you chose $d$ and $b$?

Solution. We encode the numbers in $A$ using digits from the set $\{0, 1, \ldots, n-1\}$, i.e., we set $b = n - 1$. To be able to encode all numbers in the range $\{0, 1, \ldots, n^c\}$ it is required that $(b + 1)^d \geq n^c + 1$ (we can encode $(b + 1)^d$ different numbers using $d$ digits where each digit comes from a set of cardinality $b + 1$, and the cardinality of the set $\{0, 1, \ldots, n^c\}$ is $n^c + 1$). Since $(b + 1)^d = n^d$, we can set $d = c + 1$, since

$$n^{c+1} \geq n^c + 1$$

holds for every $n \geq 2$ (assuming that $c \geq 1$). The runtime then is

$$O(d(n + b)) = O((c + 1)(n + (n - 1))) = O((c + 1)2n) = O(n),$$

since 2 and $c + 1$ are both constants.

5 Loop Invariant for Radixsort

Radixsort is defined as follows:

\begin{verbatim}
Require: Array $A$ of length $n$ consisting of $d$-digit numbers where each digit is taken from the set $\{0, 1, \ldots, b\}$
1: for $i = 1, \ldots, d$ do
2:   Use a stable sort algorithm to sort array $A$ on digit $i$
3: end for
\end{verbatim}

(least significant digit is digit 1)

In this exercise we prove correctness of Radixsort via the following loop invariant:

At the beginning of iteration $i$ of the for-loop, i.e., after $i$ has been updated in Line 1 but Line 2 has not yet been executed, the following holds:

The integers in $A$ are sorted with respect to their last $i - 1$ digits.

1. Initialization: Argue that the loop-invariant holds for $i = 1$.

Solution. In the beginning of the iteration with $i = 1$ the loop-invariant states that the integers in $A$ are sorted with respect to their last $i - 1 = 0$ digits. This is trivially true.

2. Maintenance: Suppose that the loop-invariant is true for some $i$. Show that it then also holds for $i + 1$.

Hint: You need to use the fact that the employed sorting algorithm as a subroutine is stable.
Solution. Suppose that the integers in $A$ are sorted with respect to their last $i-1$ digits at the beginning of iteration $i$. We will show that at the beginning of iteration $i+1$ the integers are sorted with respect to their last $i$ digits.

Let $A_{i+1}$ be the state of $A$ in the beginning of iteration $i+1$. For an integer $x$, let $x^{(i)}$ be the integer obtained by removing all but the last $i$ digits from $x$. Suppose for the sake of a contradiction that there are indices $j, k$ with $j < k$ such that $(A_{i+1}[j])^{(i)} > (A_{i+1}[k])^{(i)}$. If such integers exist then the loop invariant would not hold. We will show that assuming that these integers exist leads to a contradiction.

First, suppose that digit $i$ of $(A_{i+1}[j])^{(i)}$ and digit $i$ of $(A_{i+1}[k])^{(i)}$ are identical. Note that this implies $(A_{i+1}[j])^{(i-1)} > (A_{i+1}[k])^{(i-1)}$. Observe that in iteration $i$, the digits are sorted with respect to digit $i$. Since the subroutine employed in Radixsort is a stable sort algorithm, the relative order of the two numbers has not changed since their $i$th digits are identical. This implies that the relative order of the two numbers was the same at the beginning of iteration $i$. This is a contradiction, since the loop invariant at the beginning of iteration $i$ states that the digits are sorted with respect to their $i-1$ last digits, however, $(A_{i+1}[j])^{(i-1)} > (A_{i+1}[k])^{(i-1)}$ holds.

Next, suppose that digit $i$ of $(A_{i+1}[j])^{(i)}$ and digit $i$ of $(A_{i+1}[k])^{(i)}$ are different. Then, since $(A_{i+1}[j])^{(i)} > (A_{i+1}[k])^{(i)}$ we have that digit $i$ of $(A_{i+1}[j])^{(i)}$ is necessarily larger than digit $i$ of $(A_{i+1}[k])^{(i)}$. This however is a contradiction to the fact that the numbers were sorted with respect to their $i$th digit in iteration $i$.

Hence, the assumption that there are indices $j, k$ such that $(A_{i+1}[j])^{(i)} > (A_{i+1}[k])^{(i)}$ is wrong. If no such indices exist then the integers in $A$ are sorted with respect to their last $i$ digits at the beginning of iteration $i+1$.

3. Termination: Use the loop-invariant to conclude that $A$ is sorted after the execution of the algorithm.

Solution. After iteration $d$ (or before iteration $d+1$, which is never executed), the invariant states that the numbers in $A$ are sorted with respect to their last $d$ digits, which simply means that all numbers are now sorted with regards to all their digits.