Universität Siegen
Lehrstuhl Theoretische Informatik
Markus Lohrey

Exercise 4

Task 1
Consider the following coding functions (slides 76 and 78):

$$c_1(a_1 \dots a_t) = a_1 0 a_2 0 \dots a_{t-1} 0 a_t 1$$

and

$$c_2'(a_1 \dots a_t) = c_1(\text{bin}(\lceil \log_2(n) \rceil - t)) a_1 \dots a_t$$

for bitstrings $a_1 \dots a_t \in \{0,1\}^*$.
For an input list of length $n = 6$ we get the following code of the sink paths after applying Heapsort:

$$100101001111101001110010100111 0010$$

For all sink paths c_2' is used. What is the input list?

Solution
Let $f(t) := \text{bin}(\lceil \log_2(n) \rceil - t)$. For $n = 6$ we have $\lceil \log_2(n) \rceil = 3$ and hence

t	1	2	3
$f(t)$	10	1	0
$c_1(f(t))$	1001	11	01

Now we scan the string from left to right and search for the substrings of the last row. If there is a match, the next t bits encode a sink path. This yields

$$1001 \quad 0 \quad 1001 \quad 1 \quad 11 \quad 10 \quad 1001 \quad 1 \quad 1001 \quad 0 \quad 1001 \quad 1 \quad 1001 \quad 0.$$

We can assume that the sorted list is $[1,2,3,4,5,6]$. In order to obtain the list before applying Heapsort, we need to do the operations in the opposite order, while scanning the string (only the sink paths now) from right to left. Also we know that the elements were exactly picked in the order 6,5,4,3,2,1. The last 3 sink paths correspond to the build-heap procedure. Recall: 0 = left, 1 = right.

1. $[2,1] \to [1,2]$
2. $[3,2,1] \to [1,2,3]$
3. $[4,2,3,1] \to [2,4,3,1]$
4. $[5,4,3,1,2] \to [3,4,5,1,2]$
5. $[6,4,5,1,2,3] \to [3,4,6,1,2,5]$
6. $[3,2,6,1,4,5]$
7. $[3,2,5,1,4,6]$ (this is the answer)

Task 2
Is there a comparison-based sorting algorithm and a number $c > 0$ such that the following holds: The proportion of all input lists of length n on which the algorithm makes at most $c \cdot n$ comparisons is at least $\frac{1}{2^n}$.

Solution
No: A binary tree of height h has at most $1 + 2 + 4 + \cdots + 2^h = 2^{h+1} - 1$ many nodes. Hence, there exist at most $2^{c \cdot n+1} - 1$ many lists, where the algorithm makes at most $c \cdot n$ many comparisons (height = number of comparisons needed; number of leaves $\leq$ number of nodes). This means, regardless of the algorithm, the proportion of these lists is at most $\frac{2^{c \cdot n+1}-1}{n!}$. But we have

$$\frac{2^{c \cdot n+1} - 1}{n!} \geq \frac{1}{2^n} \iff 2^{(c+1) \cdot n+1} - 2^n \geq n!,$$

which is wrong for large enough $n \in \mathbb{N}$. We have $2^{d \cdot n} \in o(n!)$.

Task 3
Sort the following list via Radixsort.

$$[331, 489, 635, 320, 759, 425, 185, 920]$$

Solution
In round i we sort the array with respect to the i-th digit (from the least to the most significant) with Countingsort:

1.
   - $**0$: 2 elements ($[320, 920]$)
   - $**1$: 1 element ($[331]$)
   - $**5$: 3 elements ($[635, 425, 185]$)
   - $**9$: 2 elements ($[489, 759]$)

   $\Rightarrow [320, 920, 331, 635, 425, 185, 489, 759]$

2.
   - $*2*$: 3 elements ($[320, 920, 425]$)
   - $*3*$: 2 elements ($[331, 635]$)
   - $*5*$: 1 element ($[795]$)
   - $*8*$: 2 elements ($[185, 489]$)

   $\Rightarrow [320, 920, 425, 331, 635, 795, 185, 489]$

3.
- $1**$: 1 element ($[185]$)
- $3**$: 2 elements ($[320, 331]$)
- $4**$: 2 elements ($[425, 489]$)
- $6**$: 1 element ($[635]$)
- $7**$: 1 element ($[795]$)
- $9**$: 1 element ($[920]$)

$\Rightarrow [185, 320, 331, 425, 489, 635, 795, 920]$

Task 4
Sort the following list via Bucketsort.

$$[0.22, 0.87, 0.41, 0.05, 0.37, 0.84, 0.59, 0.28, 0.85, 0.33]$$

You can sort each bucket by using a blackbox (an arbitrary sorting algorithm).

Solution
Step 1: We create 10 buckets, since the array has length 10. This means, the elements are sorted by their first decimal place in this case.

$B[0]$	$B[1]$	$B[2]$	$B[3]$	$B[4]$	$B[5]$	$B[6]$	$B[7]$	$B[8]$	$B[9]$
$[0.05]$		$[0.22, 0.28]$	$[0.37, 0.33]$	$[0.41]$	$[0.59]$			$[0.87, 0.84, 0.85]$	

Step 2: We sort each bucket by using an arbitrary sorting algorithm:

$B[0]$	$B[1]$	$B[2]$	$B[3]$	$B[4]$	$B[5]$	$B[6]$	$B[7]$	$B[8]$	$B[9]$
$[0.05]$		$[0.22, 0.28]$	$[0.33, 0.37]$	$[0.41]$	$[0.59]$			$[0.84, 0.85, 0.87]$	

Step 3: Append the lists into a single list. This yields

$$[0.05, 0.22, 0.28, 0.33, 0.37, 0.41, 0.59, 0.84, 0.85, 0.87].$$

Task 5
Show that the median of five numbers can be computed using six comparisons.

Solution
The idea is to find two elements of which we know that at least three other elements are greater. Such an element cannot be the median. After this we have three elements remaining. We then have to find out which element among these is the smallest, which is the median.
Let a, b, c, d, e be the five numbers. We assume WLOG that $a < b < c < d < e$ but we have to be careful to discover this through comparisons.

- Compare a and b. Also compare c and d. We now know that $a < b$ and $c < d$.
- Compare the smaller elements of $\{a, b\}$ and $\{c, d\}$ which are a and c. We now know that $a < c$. Since we know that b, c and d are greater than a, it cannot be the median.

- Compare b and e, so we now know that $b < e$.
- Compare the smaller elements of $\{b, e\}$ and $\{c, d\}$ which are b and c. We now know that $b < c$. Since we know that c, d and e are greater than b, it cannot be the median.
- We are now left with $\{c, d, e\}$. The smallest element among these three is the median. We already know that $c < d$, so d cannot be the median. We compare c and e and find out that $c < e$. Therefore, c is the smallest element of $\{c, d, e\}$ and thus is the median.

Task 6
Let $(x_1, y_1), \ldots, (x_n, y_n)$ be n points in the plane $\mathbb{R}^2$. Find a line g parallel to the x-axis in time $\mathcal{O}(n)$, such that the sum of the distances between g and the points is minimal. Prove that your line is indeed optimal.

Solution
The x-values can be ignored for this task.
If n is odd, then g is optimal, if it is passing through the median y_{med}: Assume that we are moving g up by $\varepsilon > 0$. Then for at least $\lceil \frac{n}{2} \rceil$ many points with $y_i \leq y_{\text{med}}$ the distance to g is increasing by ε. But only for at most $\lfloor \frac{n}{2} \rfloor$ many points with $y_i > y_{\text{med}}$ the distance to g is decreasing by ε. The same argument is true for shifting the line down.
If n is even, every line between the two medians is optimal.
Hence the problem can be solved in linear time by finding the median of the y-values.

Solution (Bonus - Overview Sorting Algorithms)
Countingsort, Radixsort and Bucketsort are not in the list, since they are not comparison based. With certain assumptions, Radixsort and Bucketsort run in linear time.

Name	Best	Average	Worst	In-Place?	Stable? *
Mergesort	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	No	Yes
Quicksort	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n^2)$	Yes	No
Heapsort	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	Yes	No
Bubblesort	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	Yes	Yes

*This means that if $A[i] = A[j]$ for $i < j$,
then in the sorted array the entry of $A[i]$ is left of $A[j]$.