Problem 1.1

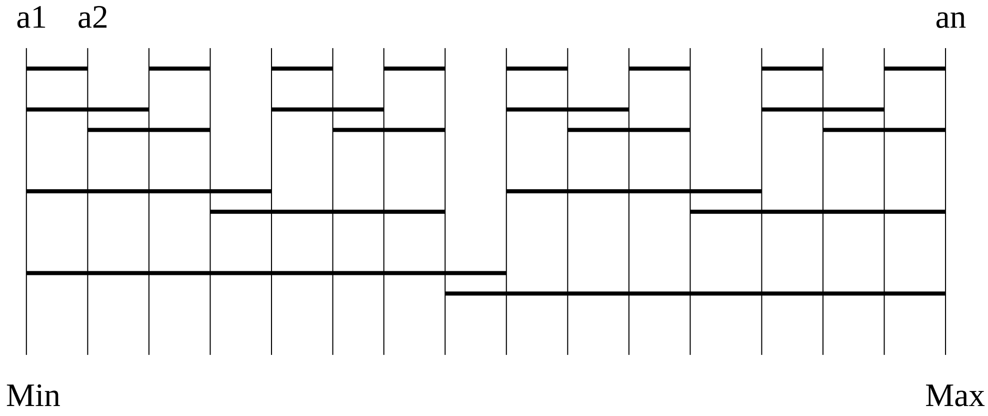

Figure 1: NC circuit of comparators which outputs min, max

Input to this circuit is a vector of integers $a_1, \ldots, a_n$ Gates used in this circuit are comparators.

The idea is to first compute the minimum / maximum for 2 elements, in the second step we compute the minimum / maximum of 4 elements, in the third step we compute the minimum / maximum of 8 elements, etc.

In the first step we compare two elements which are next to each other with each other. We know that at position a_i we have the $min(a_i, a_{i+1})$ and at position a_{i+1} we will find the $max(a_i, a_{i+1})$. In the second step we compare the minima of the operation $compare(a_i, a_{i+1})$ and $compare(a_{i+2}, a_{i+3})$ with each other and the maxima of $compare(a_i, a_{i+1})$ and $compare(a_{i+2}, a_{i+3})$, i.e. $min(a_i, a_{i+2})$ and $max(a_{i+1}), a_{i+3})$.

In the k-th step we compare the minima and maxima from the $k-1$-th step with each other, i.e. $min(a_i, a_{i+2^{k-1}+1})$ and the $max(a_{i+2^{k-1}}, a_{i+2^k})$

We will argue in Problem 1.3 why this circuit uses a minimal number of gates and computes its output fastest.

Problem 1.2

Time Analysis: We can compute the output in $\log n$ time.

Gate Analysis:

$$
\begin{aligned}
G(n) &= \frac{n}{2} + 2 * \frac{n}{4} + 2 * \frac{n}{8} + 2 * \frac{n}{16} \dots \\
&= \frac{n}{2} + \frac{n}{2} + \frac{n}{4} + \frac{n}{8} \dots \\
&= \frac{n}{2} + n * \sum_{k=1}^{(\log n)-1} (\frac{1}{2})^k \\
&= \frac{n}{2} - n + n * \sum_{k=0}^{(\log n)-1} (\frac{1}{2})^k \\
&= \frac{n}{2} - n + 2 * n - 2 = \frac{3}{2} n - 2
\end{aligned}
$$

Problem 1.3

This analysis is based on the comparator gate which takes two inputs.

Time Analysis: The optimal solution to compute the minimum of n inputs takes $\log n$ steps. The number of steps is measured by the depth of the circuit. In order to output the minimum in the last step (k-th step), we had to compare 2 elements in the $(k-1)$ step using the comparator gate. So, in the $(k-2)$ th step, we compared 4 elements using the comparator gates. In the first step, we therefore are able to compare 2^k elements. If n is the total number of inputs (2^k), then we need $\log n$ steps to find the minimum.

The optimal solution to compute the maximum of n inputs takes also $\log n$ steps by the same argument.

As the maximum and minimum are independent of each other, we can compute them in parallel. Hence, the optimal solution to compute the maximum and the minimum of n inputs takes $\log n$ time.

As our algorithm runs in $\log n$ time, it is optimal with respect to time.

Gate Analysis: In order to output the minimum in the k-th step (last step) we had to compare 2 elements in the $k-1$ th step using one gate. These two elements were computed in the $k-2$th step using 2 gates. We know the optimal depth of a circuit is $\log n$. Therefore, an optimal solution for finding the minimum needs in total $\sum_{i=0}^{(\log n)-1} 2^i$ gates, i.e. $n-1$ gates. An optimal solution for finding the maxima in k steps needs also $n-1$ gates. In the first step we need $\frac{n}{2}$ gates. As the comparator gate outputs minimum and maximum in one step, we can use the same gate for outputting the minima and maxima in the first step: hence we need $(n-1)+(n-1)-\frac{n}{2}$ gates in total. We can simplify $(n-1)+(n-1)-\frac{n}{2}$ to get $\frac{3}{2}n-2$. Therefore the optimal number of gates needed is $\frac{3}{2}n-2$.

As our circuit needs exactly $\frac{3}{2}n-2$ gates, it is optimal with respect to number of comparator gates needed.

Problem 2

see figure 2. Gates used are comparator gates. Input is a vector of 8 elements.

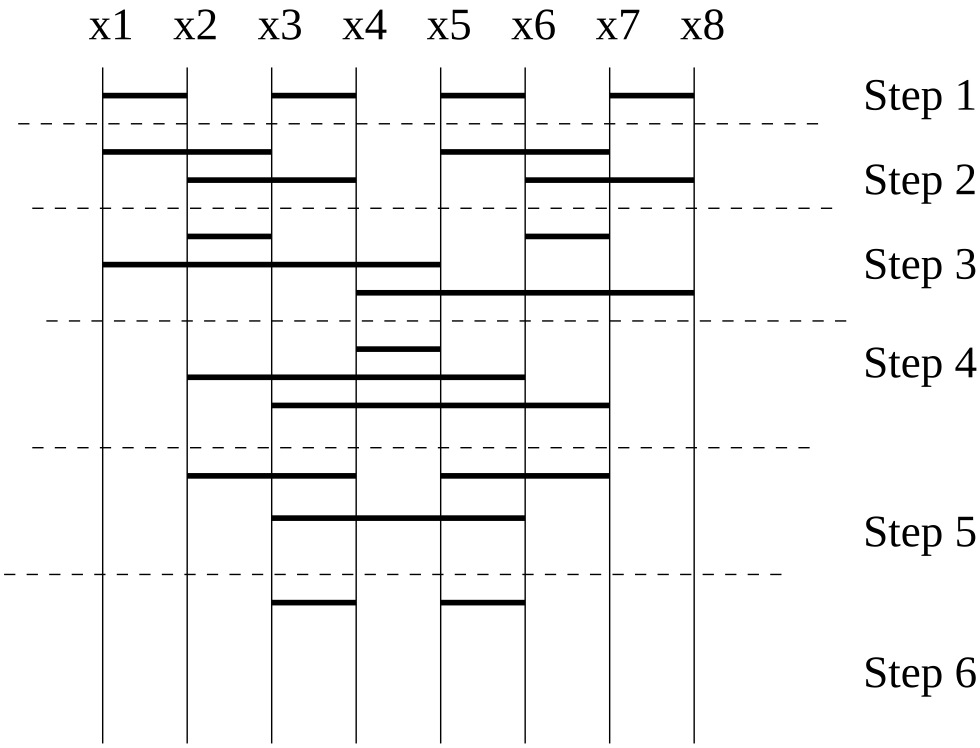

Figure 2: NC circuit using comparators for sorting of 8 elements in 6 step

Problem 3

see figure 3. Gates used are comparator gates. We extend the idea developed in Problem 1, and add extra levels of comparator gates, to ensure that the second minimum is indeed the second minimum, and the second maximum is indeed the sec. maximum.

We start by creating a 4 elements in sorted order, where the first two are the minima the last two are the maxima we are looking for. In the next step we compute the min / max pairs of 8 elements, etc.

$t(n) = 3 + 2 * (\log n - 3)$ and

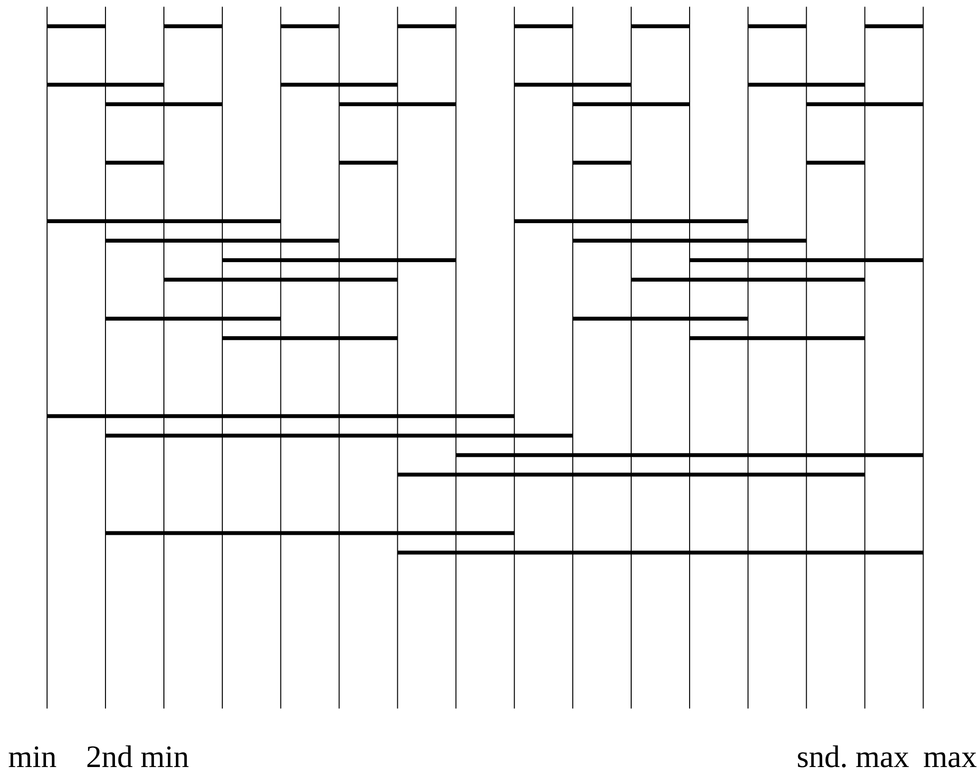

Figure 3: NC circuit of comparators which outputs min, snd. min, snd. max, max

$$
\begin{aligned}
g(n) &= \frac{n}{2} + 2 * \frac{n}{4} + \frac{n}{4} + 4\frac{n}{8} + 2\frac{n}{8} + +4\frac{n}{16} + 2\frac{n}{16} + \ldots \\
&= \frac{n}{2} + 3 * \frac{n}{4} + 6n * (\sum_{k=3}^{\log n} \frac{1}{2^i}) \\
&= \frac{n}{2} + 3 * \frac{n}{4} + 6n * (\sum_{k=0}^{\log n} \frac{1}{2^i} - 1 - \frac{1}{2} - \frac{1}{4}) \\
&= \frac{n}{2} + 3 * \frac{n}{4} + 6n * (-\frac{1}{n} + 2 - 1\frac{3}{4}) \\
&= \frac{n}{2} + 3 * \frac{n}{4} + 6n * (-\frac{1}{n} + \frac{1}{4}) \\
&= \frac{n}{2} + 3 * \frac{n}{4} - 6 + \frac{6}{4}n) \\
&= \frac{n}{2} + 3 * \frac{n}{4} - 6 + \frac{3}{2}n) \\
&= 2\frac{3}{4}n - 6
\end{aligned}
$$

As the circuit runs in $\log n$ time and uses polynomial number of gates, the circuit is in NC.

Problem 5

Part A

By the access lemma, the cost of accessing element x_i is :

$$\#rot = 3(r(t) - r(x_i)) + 1$$

In order to get a tight bound we assign nodes which are accessed frequently (i.e. the probability p_i is high) a high weight. Let the weight of node x_i be the assigned probability to access node x_i, i.e. $w(x_i) = p_i$. As all the probabilities sum up to one, we know that the size of the whole tree with root t must be 1, i.e. $s(t) = \sum_{y \in t} w(y) = 1$; therefore the rank $r(t) = \log s(t) = \log 1 = 0$.

$$\begin{aligned}
3(r(t) - r(x_i)) + 1 &= 3(\log \sum_{y \in t} w(y) - \log \sum_{y \in x} w(y)) + 1 \\
&= -3(\log \sum_{y \in x} w(y)) + 1 \\
&= 3(\log \frac{1}{\sum_{y \in x} w(y)}) + 1 \\
&\leq 3(\log \frac{1}{w(x)}) + 1 \\
&= 3(\log \frac{1}{p_x}) + 1
\end{aligned}$$

The expected cost is then the following sum:

$$\begin{aligned}
\#rot &= 1 + 3\sum_{x=1}^{n} p(x) \log \frac{1}{p_x} \\
&= 1 + 3\sum_{x=1}^{n-1} p(x) \log 2^x + 3 * \frac{1}{2^{n-1}} * \log 2^{n-1} \\
&= 1 + \frac{3(n-1)}{2^{n-1}} + 3\sum_{x=1}^{n-1} x(\frac{1}{2})^x \\
&= 1 + \frac{3(n-1)}{2^{n-1}} + 3 * 4((\frac{1}{2})^n - 1) \\
&= \frac{6(n-1)}{2^n} - 11
\end{aligned}$$

Part B

To access a node x in a treap,we need to search for it. We follow the path from the root down to x. The number of comparisons is the length of the path from the root to x. By theorem 1 (handout) the expected length of a path to a node x in a treap on n nodes is :

$$H(x) + H(n - x + 1) - 1$$

The expected cost is then the following sum:

$$\begin{aligned}
&\sum_{x=1}^{n} p(x) * (H(x) + H(n - x + 1) - 1) \\
&= \sum_{x=1}^{n} p(x) * H(x) + \sum_{x=1}^{n} p(x) * H(n - x - 1) - \sum_{x=1}^{n} p(x) \\
&= 2 * \sum_{x=1}^{n} p(x) * H(x) - \sum_{x=1}^{n} p(x)
\end{aligned}$$

Part C

$$\begin{aligned}
&2 * \sum_{x=1}^{n} p(x) * H(x) - \sum_{x=1}^{n} p(x) \\
&= 2 * \sum_{x=1}^{n-1} p(x) * (H(x) - 1) + p(n) * (H(n) - 1) \\
&= 2 * \sum_{x=1}^{n-1} (\frac{1}{2})^x (H(x) - 1) + \frac{1}{2^n}(H(n) - 1) \\
&\leq 2 * \sum_{x=1}^{n-1} (\frac{1}{2})^x (H(n) - 1) + \frac{1}{2^n}(H(n) - 1) \\
&= 2 * (H(n) - 1) * \sum_{x=1}^{n-1} (\frac{1}{2})^x + \frac{1}{2^n}(H(n) - 1) \\
&= 2 * (H(n) - 1) * (\sum_{x=0}^{n-1} (\frac{1}{2})^x - 1) + \frac{1}{2^n}(H(n) - 1) \\
&= 2 * (H(n) - 1) * (1 - \frac{1}{2^{n-2}}) + \frac{1}{2^n}(H(n) - 1) \\
&= (H(n) - 1) * (2 - \frac{1}{2^{n-3}} + \frac{1}{2^n}) \\
&= (H(n) - 1) * (2 - \frac{2^3}{2^n} + \frac{1}{2^n}) \\
&\leq 2 * (H(n) - 1)
\end{aligned}$$

Note the last approximation, that $\frac{1}{2^n} < \frac{2^3}{2^n}$ is only valid, for $n > 3$.

Problem 6

Part A

Let G be a connected undirected graph with n vertices and n edges. To connect n vertices we need at least $n-1$ edges; so the additional edge available will lead to one cycle in the graph. Let G' be the connected graph with n vertices and $n-1$ edges, i.e. G' is the graph G with the edge, which causes the cycle, removed.

We will show that there exists a 2-vertex $\frac{1}{3}$-$\frac{2}{3}$ separator in the Graph G'. This separator is also a separator for the graph G, as the number of vertices does not change.

Pick any vertex s in G'. Then we can view the graph G' as a tree with root node s and subtrees $T_1, T_2, \ldots T_k$.

Without loss of generality, we assume that the size T_1 is greater than the size of the rest, i.e. $\sum_{2 \leq i \leq k} size(T_i) < size(T_1)$. (Otherwise we can consider $T_2 \ldots T_n$ versus T_1.)

If the size of T_1 is less or equal than $\frac{2}{3}n$ ($size(T_1) \leq \frac{2}{3}n$) and the size of the rest of the subtrees $\sum_{2 \leq i \leq k} size(T_i) \leq \frac{1}{3}n$ then we found a separator; pick as a separator the root node s and the root of T_1.

If the size of T_1 is greater than $\frac{2}{3}n$ ($size(T_1) \geq \frac{2}{3}n$), then we need to make the subtree smaller in order to find a $\frac{1}{3}$-$\frac{2}{3}$ separator.

T_1 has again several subtrees $T_{2,1}, \ldots T_{2,l}$. At depth d we will have subtrees $T_{d,1}, \ldots T_{d,l}$. We recursively descend the tree T_1 until the size of the subtree $T_{d,1}$ is less or equal than $\frac{2}{3}n$. Then we know $T_{d,1} \leq \frac{2}{3}n$ and all the nodes which are to the right, i.e. $\sum_{2 \leq i \leq k} size(T_{1,i}) + \sum_{2 \leq i \leq k} size(T_{2,i}) + \ldots + \sum_{2 \leq i \leq k} size(T_{d,i})$ is less than $\frac{1}{3}n$.

And we can take root of $T_{d,1}$ together with its ancestor, as a separator.

Part B

First, construct a graph G' which does not have a cycle and remove the edge which causes the cycle. This can be done by breadth first search, in at most n steps. In the worst case we will have a cycle of n nodes. In this case we need n steps to detect the cycle.

```
removeCycle G
. Start: Pick any vertex s, add s to a book-keeping list
   (= vertices which we already looked at)
. Compute by breadth first search on s the neighbors of s.
   Let the neighbors of s be n1 ... nk.
      if ni already occurs in the book-keeping list,
        then we have a cycle and we remove the edge (s, ni)
      otherwise we again compute for all i all neighbors of ni
      we repeat this process until there are no vertices left
      or until we found a cycle.
```

Assume we already have a graph G' which does not have a cycle.

```
. Pick any vertex s of graph G' and do breadth first search on s
   to get all subtrees of s.
. find separator (T1, [s, T2 ... Tn])
     if size(T1) < size ([s, T2 ... Tn]) then
       separator ([T2 ... Tn], [s, T1])
     else
       if size(T1) <= 2/3 n then
         % size([T2, ..., Tn]) < 1/3 n
         return s and the root of T1 as a separator
         we are done
       else
         % size (T1) > 2/3 n
         find the leftmost tree which is stored in T1 by depth first search.
         let the leftmost tree be T1'
         let s' be the root of T1'
         get all subtrees S1 ... Sn from s'
         find separator (S1, [s', S2 ... Sm, s, T2, ..., Tn])
```

The algorithm will run in linear time. It takes linear time to remove the cycle from the graph. It also takes linear time to find the separator. For finding the separator, the worst case is when the tree is a chain. Then we will descend this chain $\frac{1}{3}n$ times.