5.1 Quick sort

The quick sort algorithm is described in the following.

$$
\begin{array}{lcl}
\texttt{quickSort} & :: & \text{Ord}\ \alpha \Rightarrow [\alpha] \to [\alpha] \\
\texttt{quickSort}\ [\,] & = & [\,] \\
\texttt{quickSort}\ xs & = & (\texttt{quickSort}\ A) \mathbin{+\!\!+} C \mathbin{+\!\!+} (\texttt{quickSort}\ B) \\
\quad \textbf{where} & & \begin{array}{lcl} pv & = & \texttt{pivot}\ xs \\ A & = & \texttt{filter}\ (<\ pv)\ xs \\ B & = & \texttt{filter}\ (>\ pv)\ xs \\ C & = & \texttt{filter}\ (=\ pv)\ xs \end{array}
\end{array}
$$

Theorem 1. *The average and worse time complexity is $\mathcal{O}(n \log n)$ which the later required the median of medians (See Section 5.4.2).*

Proof. Since with the median of medians, finding a median cost $\mathcal{O}(n)$. And if we use median as the pivot, the recursive formula is

$$T(n) = 2T(n/2) + \mathcal{O}(n) \implies T(n) \in \mathcal{O}(n).$$

□

5.2 Merge sort

The merge sort algorithm is described in the following.

$$
\begin{array}{lcl}
\texttt{mergeSort} & :: & \text{Ord}\ \alpha \Rightarrow [\alpha] \to [\alpha] \\
\texttt{mergeSort}\ [\,] & = & [\,] \\
\texttt{mergeSort}\ [x] & = & [x] \\
\texttt{mergeSort}\ xs & = & \texttt{mergeTwoSorted}\ (\texttt{firstHalf}\ xs)\ (\texttt{secondHalf}\ xs)
\end{array}
$$

Where `firstHalf` and `secondHalf` cut the list into first $\lfloor n/2 \rfloor$ and the remains. Now if we could preform `mergeTwoSorted` in $\mathcal{O}(n)$, then the time complexity is $T(n) = 2T(n/2) + n$ which yields $T(n) = \mathcal{O}(n \log n)$.

Actually, there is a clever way to do it.

$$
\begin{array}{lcl}
\texttt{mergeTwoSorted} & :: & \text{Ord}\ \alpha \Rightarrow [\alpha] \to [\alpha] \to [\alpha] \\
\texttt{mergeTwoSorted}\ [\,]\ ys & = & ys \\
\texttt{mergeTwoSorted}\ xs\ [\,] & = & xs \\
\texttt{mergeTwoSorted}\ (x : xs)\ (y : ys) & & \\
\qquad\qquad \mid x \leq y & = & x : \texttt{mergeTwoSorted}\ xs\ (y : ys) \\
\qquad\qquad \mid \textbf{otherwise} & = & y : \texttt{mergeTwoSorted}\ (x : xs)\ ys
\end{array}
$$

5.3 Time complexity of comparison sorting

Theorem 2. *For comparison algorithms (i.e. algorithms which only assume the elements in the list is comparable), the average time complexity has a lower bound $\mathcal{O}(n \log n)$.*

Proof. Assuming that every elements in the list is different.

Let P be the original list, and P' be the list after sorting. To sort the list is equal to decide the permutation σ such that $P' = \sigma P$. There are $n!$ of such permutation, but exactly one satisfied the equality.

Every time, we could only compare two elements x, y and get two outcomes, either $x > y$ or $x < y$. If we do comparison m times, there are 2^m different outcomes. Base on these information, we have to decide σ. That is, we could imagine that our algorithm is a function $f = (\alpha_1, \alpha_2, \cdots, \alpha_m) \to \sigma$ where α_i is the results of the i-th comparison. The domain of a function must be larger than the image, so $2^m \geq n! \implies m \geq \mathcal{O}(n \log n)$ □

5.4 Order Statistics

5.4.1 k-th element

The following function $\texttt{select}(xs, k)$ returns the k-th element (in 0-base) in the list xs.

$$
\begin{array}{lll}
\texttt{select} & :: & \text{Ord } \alpha \Rightarrow [\alpha] \to \mathbb{N} \to \alpha \\
\texttt{select } [x] \ 1 & = & x \\
\texttt{select } xs \ k & & \\
\quad \mid k < m & = & \texttt{select } A \ k \\
\quad \mid \textbf{otherwise} & = & \texttt{select } B \ (k - m) \\
\quad \textbf{where} & & pv = \texttt{pivot } xs \\
 & & A = \texttt{filter } (< pv) \ xs \\
 & & B = \texttt{filter } (\geq pv) \ xs \\
 & & m = \texttt{length } A
\end{array}
$$

Theorem 3. *If* `pivot` *randomly choose a pivot, the algorithm above has average time complexity $\mathcal{O}(n)$, where n is the length of the list.*

Proof. The recursive formula is

$$T(n) = E[T(\max(|A|, |B|))] + n = \left(\frac{1}{n} \sum_{m=1}^{n-1} T(\max(m, n-m-1)) \right) + n$$

Assume that $T(k) \leq ck$ for some constant c for all $k < n$. Then

$$
\begin{aligned}
T(n) &= \left(\frac{1}{n} \sum_{m=1}^{n-1} T(\max(m, n-m-1)) \right) + n \approx \left(\frac{2}{n} \sum_{m=\lceil n/2 \rceil}^{n-1} T(m) \right) + n \\
&\leq \left(\frac{2}{n} \sum_{m=\lceil n/2 \rceil}^{n-1} cm \right) + n \leq \frac{2}{n} \frac{3cn^2}{8} + n = \frac{3c+4}{4} n
\end{aligned}
$$

Choose $c \geq 4$ and hence $(3c+4)/4 \leq c$ and by induction the proof is complete. □

5.4.2 Median of medians

If we slightly change how we choose the pivot in the algorithm to

$$
\begin{array}{lll}
\texttt{select}\ xs\ k & = & x \\
\quad k < m & = & \texttt{select}\ A\ k \\
\quad \textbf{otherwise} & = & \texttt{select}\ B\ (k-m) \\
\quad \textbf{where} & & \begin{array}{lll} mds & = & \texttt{map}\ \texttt{getMedianOf5}\ (\texttt{chunksOf}\ 5\ xs) \\ pv & = & \texttt{select}\ xs\ \lfloor(\texttt{length}\ mds)/2\rfloor \\ A & = & \texttt{filter}\ (<\ pv)\ xs \\ B & = & \texttt{filter}\ (\geq\ pv)\ xs \\ m & = & \texttt{length}\ A \end{array}
\end{array}
$$

Where ($\texttt{chunksOf}$ 5) groups every five elements into a chunk. The method is so called "Median of medians".

Theorem 4.

1. *pv would be greater than at least* $1/4$ *elements, and less than at least* $1/4$ *elements in mds.*
2. *The modified algorithm has a worse time complexity* $\mathcal{O}(n)$.

Proof. The length of *mds* is $\lfloor n/2 \rfloor$. Since *pv* is the median of *mds*, *pv* is greater than $\lfloor n/10 \rfloor$ elements in *mds*. Since these element are the median in the chunk it belongs, each of them is not less than 3 elements in its chunk, and hence *pv* is greater than $\lfloor n/10 \rfloor \cdot 3 \geq n/4$ elements. Similarly *pv* is less than $n/4$ elements.

The recursive formula of $T(n)$ in the worse case is

$$T(n) = T(n/5) + T(3n/4) + n$$

Assume that $T(k) \leq ck$ for some constant c for all $k < n$. Then

$$T(n) \leq \frac{cn}{5} + \frac{3cn}{4} + n = \frac{19c + 20}{20} n$$

Choose $c \geq 20$ and then $(19c + 20)/20 \leq c$. By induction the proof is complete.

□