Median and order statistics

We have input numbers $A[0], \ldots, A[n-1]$ and after sorting they would form sequence $S[0], \ldots, S[n-1]$. We want to know $S[k]$, order statistic $k$ of the input.

This problem is interesting because it is useful, AND because we can compute $S[k]$ faster than using sorting.

The idea is that we can use a linear time effort to reduce the problem size by a constant. This can be done deterministically, but it is quite a bit more complicated and slower than a method using random choices.

The simplest idea that works is to run Quick sort, but sort only as much as to assure that $A[k]$ will be in the correct position.

```c
void QuickStat (A,i,j,k) // sector A[i, j-1], stat k
{
    a = Partition (A,i,j);
    if (k < a)
        QuickStat (A,i,a,k)
    else if (k > a)
        QuickStat (A,a+1,j,k)
}
```

After $\text{QuickStat}(A,0,n,k)$ the array $A[n]$ is partially sorted, so $A[k]$ is in the correct place. One can see that on the average, if we make a recursive call, the number of keys in the sector goes down by a factor of $3/4$. Thus we get a recurrence

$$T(n) = T(0.75n) + n$$

After $\text{QuickStat}(A,0,n,k)$ the array $A[n]$ is partially sorted, so $A[k]$ is in the correct place. One can see that on the average, if we make a recursive call, the number of keys in the sector goes down by a factor of $3/4$. Thus we get a recurrence

$$T(n) = T(0.75n) + n$$
**Transform and conquer**

In divide and conquer and diminish and conquer we solve an instance of a problem using one or more solutions to smaller instances. This is recursive design.

But for a recursive design to work we often have to transform the problem.

Example 1: largest sum of a contiguous array fragment.

Instance: \( A[n] \)

Subproblem data: sector \((A, 0, i), i \leq n\).

Transformed problem:
the largest sum inside the sector and the largest sum at the end of the sector.

Now we can obtain solution for data \((A, 0, i + 1)\) from the solution for \((A, 0, i)\) in few steps.

We have several types of transformations.

We can generalize the problem, e.g. we want to find the median, but we have a method for every order statistic.

We can enhance the problem, compute some extra, as in Example 1.

We can put data in some kind of data structure that allows to quickly answer so-called queries.

Presorting of the input numbers allows to answer:

- given \( A[i] \), what is the smallest number in the set that is larger? answer in \( O(1) \) time
- given \( x \), what is the smallest number in the set that is larger? answer in \( O(\log n) \) time
- given \( x \), is \( x \) in the set?

There are many data structures, but the above questions are most basic. In data structure we not only have queries, but we can also have updates, when the set changes. And the set of the structure can include different kinds of objects and relationships.
Gaussian Elimination

Sometimes we can formulate the problem as the transformation of the input. For example, we have a set of equations:

\[
\begin{align*}
    a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\
    a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_1 \\
    \vdots \\
    a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_1
\end{align*}
\]

and we want to know the solution, if one exists. The solution is also a system of equations:

\[
\begin{align*}
    x_1 &= s_1 \\
    x_2 &= s_2 \\
    \vdots \\
    x_n &= s_n
\end{align*}
\]

thus we can view finding the solution as the transformation of the input system. We can write such a system of equations as a single matrix/vector equation

\[
Ax^T = b^T
\]

and the desired form is that \( A \) is a matrix with 1’s on the diagonal and zeros outside the diagonal.

Why are we allowed to transform matrix \( A \)? Because we can obtain a matrix that has the same solution.

When a transformation can be viewed as progress? It will be easier to draw the matrices on the board.

Next few lectures: ordered dictionary data structure.

Tonight: homework 5.