2/23

Algorithms involving sequences and sets:

Usually the input is a finite set/sequence:

Differences { 1- in sequences the order of elements is important whereas in sets it is not
2- in sets we assume an element does not appear more than once (not in multisets though) but no such assumption for sequences.

The input is always a sequence though if the order is not important we call it a set.
In this chap the input is an array of size n. Also the elements can be compared.
Binary search and sorting are very known important and universally applicable algorithms.

First Binary search:

Basic idea: cut the search space in half (or approximately so) by asking only one question

The problem: let $x_1, x_2, \ldots x_n$ be a sequence of real numbers such that $x_1 \leq x_2 \leq \ldots \leq x_n$. Given a real number z, we want to find whether z appears in the sequence and if it does, to find an index i such that $x_i = z$.

Say there is only one index i such that $x_i = z$ (in general, it might not be the case and we want to find all or the smallest and the largest one only)

compare z with $x_{\lceil \frac{n}{2} \rceil}$, if $z < x_{\lceil \frac{n}{2} \rceil}$ then z is clearly in the first half of the sequence otherwise z is in the second half. Finding z in either half is a problem of size $\frac{n}{2}$, which can be solved by induction. We handle the base case of $n=1$ by directly comparing z to the element.

```
Algorithm Binary_search(z, L, R, X)
   input: array X and integers z, L, R
   output: an index i such that X[i] = z or -1 otherwise
begin
   if L = R then
      if X[L] = z then find = L
      else Find = -1
   else
      M := ⌈1/2(L+R)⌉; if X[M] = z then find = M;
      if z < X[M] then Find = Binary_search(z, L, M-1, X)
      else Find = Binary-search(z, M+1, R, X)
   return Find;
end;
```

Time Complexity:

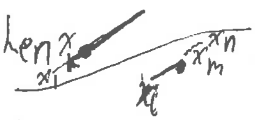
Since each time a comparison is made, the range is cut by one half the number of comparisons is $O(\log n)$. For small values of n binary search might not be as efficient as linear search.

Binary Search in a cyclic sequence

A sequence $x_1, x_2, \cdots, x_n$ is said to be cyclically sorted if the smallest number in the sequence is x_i for some unknown i, and the sequence $x_i\, x_{i+1} \cdots x_n, x_1, \cdots x_{i-1}$ is sorted in increasing order (i.e., $x_i \le x_{i+1} \le \cdots \le x_n \le x_1 \le x_2 \le \cdots x_{i-1}$)

The Problem: Given a cyclically sorted list, find the position of the minimal element in the list.

We use the idea of eliminating half of the sequence by one comparison.

Take any two numbers x_k and x_m such that $k < m$. If $x_k < x_m$, then i cannot be in the range $k < i \le m$ since then  $x_m \ge x_k$ a contradiction. On the other hand if $x_k > x_m$ then i must be in the range $k < i \le m$, since the order is switched somewhere in that range. Thus with one comparison we can eliminate half elements and we can find i is $O(\log n)$

Algorithm cyclic_find(L, R, X)

```
begin
   if L=R then return L
   else M:=⌊1/2(L+R)⌋;
      if X[M] < X[R] then cyclicfind(L, M, X)
      else cyclic_find(M+1, R, X).
```

Binary Search for a special (fixed) index:

The problem: Given a sorted sequence of distinct integers $a_1, a_2, \cdots a_n$ determine there is an index i such that $a_i = i$.

Again we cannot use binary search here, but the principle can be applied.

If $a_{\lceil \frac{n}{2} \rceil}$ is exactly $\lceil \frac{n}{2} \rceil$ then we are done; otherwise if it is less than $\frac{n}{2}$, since all numbers are distinct the value of $a_{\lceil \frac{n}{2} \rceil - 1}$ is less that $\lceil \frac{n}{2} \rceil - 1$ and so on. Thus no number in the first half of the sequence can satisfy the property and we can continue search the second half. The same holds if the answer is "greater than" (the algorithm is in the book)

<u>Binary search in sequences of unknown size:</u>

Sometime we use a procedure like binary search to double the search space rather than to halve it. Consider a regular search problem with unknown size. We cannot halve the search range, since we do not know the boundaries. Instead we search for an element $x_i \ge z$. If we can find x_i, we can do binary search from 1 to i. First we compare z and x_1. If $z \le x_1$ then $z = x_1$. Now by induction, we know $z > x_j$ for $j \ge 1$

If we compare z to x_{2j}, then we double the search space with one comparison. If $z \le x_{2j}$, we know $x_j \le z \le x_{2j}$ and we can find z with $O(\log j)$ additional comparisons. Overall if i the smallest index such that $x_i \ge z$, then it takes $O(\log i)$ comparisons to find an x_j such that $z \le x_j$ and another $O(\log i)$ to find i.

The same algorithm can also be used when the size of the sequence is known but we suspect i is very small. This is improvement since we have $O(\log i)$ instead of $O(\log n)$. However since it is $2\log i$, it is better that $\log n$ if $2\log i \le \log n \Rightarrow i \le \sqrt{n}$ or $i = O(\sqrt{n})$

We have also Interpolation search

Combination of binary search and linear search: It is used when during the search we find a value that is very close to the search number z then it seems more reasonable to continue the search in that "neighborhood" using linear search, for example, instead of blindly going to the next half point. E.g. consider when you open a book and search for a particular page number say 200.

See more on Interpolation search and more applications of binary search in the book.