Algorithms involving sequences and sets:

Usually the input is a finite set/sequence.

Differences \[ 1 \] in sequences the order of elements is important where as in sets it is not.

In sets we assume an element does not appear more than once (not in multiset),

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 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 of the smallest and the largest one only.)

Compare \( z \) with \( x_{\lfloor \frac{n}{2} \rfloor} \) if \( z < x_{\lfloor \frac{n}{2} \rfloor} \) 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 \( l,r,k \)

Output: an index \( i \) such that \( x[i] = z \) or -1 otherwise.

begin
  if \( l = r \) then
    if \( x[l] = z \) then \( \text{find} = l \)
    else \( \text{find} = -1 \)
  else
    \( m = \lfloor \frac{l+r}{2} \rfloor \); \( x[m] = z \) then \( \text{find} = m \);
    if \( z < x[m] \) then Binary search \((z,l,m-1,x)\)
    else \( \text{find} = \text{Binary search} (z,m+1,r,x) \)
  return \( \text{find} \)
end

Time complexity:

Since each time a comparison is made, the range is cut by one half.

The number of comparisons is \( \log n \).

For small values of \( n \) binary search might not be as efficient as linear search.
Binary Search in a Cyclic Sequence

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

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 $k < i < m$ since then $x_k < x_m < x_k$, a contradiction. On the other hand, if $x_k > x_m$, then $i$ must be in the range $k < i < m$, since the order is switched somewhere in that range. Thus, with one comparison, we can eliminate half the elements and we can find $i$.

Algorithm cyclic-find($L$,$R$,$X$)

begin
if $L = R$ then return $L$
else $M := \lfloor \frac{1}{2}(L+R) \rfloor$
if $X[M] < X[R]$ then cyclic-find($L$, $M$, $X$)
else cyclic-find($M+1$, $R$, $X$).
end

Binary Search for a Special (Fixed) Index:

The problem: Given a sorted sequence of distinct integers $a_1, a_2, \ldots$ and determining there is an index $i$ such that $a_i = L$.

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

If $\lfloor \frac{n}{2} \rfloor$ is exactly $[\frac{n}{2}]$ then we are done; otherwise if it is less than $\frac{n}{2}$, since all numbers are distinct, the value of $\lfloor \frac{n}{2} \rfloor$ is less than $[\frac{n}{2}]$ 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).

Binary Search in Sequences of Unknown Size:

Sometimwe we use a procedure like binary search to double the search space rather than to halve it. Consider the 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 $x_i \geq Z$. If we can find $x_i$, we can do binary search from $1$ to $i$.

First we compare $Z$ and $x_i$. If $Z \leq x_i$ then $Z = x_i$. Now by induction, we know...
If we compare $z$ to $x_j$, then we double the search space with one comparison.

If $z \leq x_j$, we know $x_j \leq z \leq x_i$ and we can find $z$ with $O(\log j)$ additional comparisons. Overall, if $i$ is the smallest index such that $x_i \geq z$, then it takes $O(\log j)$ comparisons to find an $x_j$ such that $z \leq x_j$ 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 $O(\log j)$ instead of $O(\log n)$.

However, since $i$ is $2 \log e$, it is better that $\log n$ if $2 \log i \leq \log n \Rightarrow i \leq e \log n$ or $i = 0(\sqrt{n})$.

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**.