CS19001: Programming and Data Structures Laboratory

Aritra Hazra;
CSE, IIT Kharagpur

http://cse.iitkgp.ac.in/~aritrah/course/lab/PDS/Autumn2019/

08-Nov-2019

Sorting

- Suppose you have an array A[ ] of n elements (say, integers). They are stored in the array locations,

$$A[0], A[1], \ldots, A[i], \ldots, A[n-1]$$

- We want to rearrange these integers in such a way that after the rearrangement, we have either of the following:

$$A[0] \leq A[1] \leq \cdots \leq A[i] \leq \cdots \leq A[n-1]$$

$$A[0] \geq A[1] \geq \cdots \geq A[i] \geq \cdots \geq A[n-1]$$

- Then, the resultant array is called **sorted** in either **ascending** or **descending** order, respectively.
- There are many such sorting methods. *Bubble-sort* and *Selection-sort* are two among them.

Bubble-sort (in ascending order)

Code

```
for (i=n-2; i>=0; --i)
{
   for (j=0; j<=i; ++j)
   {
      if (A[j] > A[j+1])
      {
         t = A[j];
         A[j] = A[j+1];
         A[j+1] = t;
      }
   }
}
```

Working Principle

A[4] = {4,3,2,1}
i,j: $A \rightarrow A'$

bubble till position i=4-2=2.
2,0: 4,3,2,1 → 3,4,2,1
2,1: 3,4,2,1 → 3,2,4,1
2,2: 3,2,4,1 → 3,2,1,4
bubble till position i=1
1,0: 3,2,1,4 → 2,3,1,4
1,1: 2,3,1,4 → 2,1,3,4
bubble till position i=0
0,0: 2,1,3,4 → 1,2,3,4

Selection-sort (in ascending order)

Code

```
for (i=n-1; i>0; --i)
{
   m = i;
   for (j=0; j<i; ++j)
   {
      if (A[j] > A[m])
         m = j;
   }
   t = A[i];
   A[i] = A[m];
   A[m] = t;
} // Why swap if i=m?
```

Working Principle

A[4] = {4,3,2,1}

i = 3 → m = 3

j = 0: m = 0

j = 1: m = 0

j = 2: m = 0

A[4] = {1,3,2,4}

i = 2 → m = 2

j = 0: m = 2

j = 1: m = 1

A[4] = {1,2,3,4}

i = 1 → m = 1

j = 0: m = 1

A[4] = {1,2,3,4}

Searching

- Suppose you have an array A[ ] of n elements (say, integers). They are stored in the array locations,

$$A[0], A[1], \ldots, A[i], \ldots, A[n-1]$$

- We want to search/report the location/index of a particular value, say v, from this array of integers.
- We report the index 'k' ($0 \leq k < n$), if $A[k] = v$. Otherwise, we may report '-1' to denote that the searched element, v, is not found.
- Given an *unordered* array, you have to compare each element of the array **sequentially** to find the index,

$$\textit{For all } i \ (0 \leq i < n), \ \textit{whether } A[i] = v?$$

- However, for *ordered* (ascending / descending) arrays, things are more exciting! *We shall study these variants.*

Searching in an Unordered Array

Forward-Iteration

```
for (i=0; i<n; ++i)
  if(A[i] == v)
    break;
// Answer: found at i
if(i == n) i = -1;
// Answer: not found
```

Backward-Iteration

```
for (i=n-1; i>=0; --i)
  if(A[i] == v)
    break;
// Answer: found at i
```

Recursive-Code

```
int seqSr ( int A[], int n,
                     int v )
{
  if (n > 0)
  {
    if(A[n-1] == v)
      return n-1;
    else
      return (seqSr(A,n-1,v));
  }
  else
  {
    return -1;
  }
}
```

Searching in an Ordered Array

Idea of Binary Search:

Consider a sorted array A and an element (say v) as input. The goal is to report whether the element is present in the array and in that case what is the corresponding array index.

- Choose the middle element $A[\frac{n}{2}]$
- If $v == A[\frac{n}{2}]$, we are done
- If $v < A[\frac{n}{2}]$, search for v between $A[0]$ and $A[\frac{n}{2} - 1]$
- If $v > A[\frac{n}{2}]$, search for v between $A[\frac{n}{2} + 1]$ and $A[n - 1]$
- Repeat until v is found or no more elements remain to be searched.

We consider three variables first, last and mid pointing to array beginning, end and middle respectively. We keep on updating these three elements in each iteration recursively.

Searching in an Ordered Array

Binary Search

```
int binSr ( int A[], int v,
            int si, int ei )
{
 int mi;
 if (si <= ei)
 {
  mi = (si+ei)/2;
  if(A[mi] > v)
   return binSr(A,v,si,mi-1);
  else if (A[mi] < v)
   return binSr(A,v,mi+1,ei);
  else
   return mi;
 }
 else
  return -1;
}
```

Working Principle

Ex-1: A[5] = {1, 2, 3, 4, 5}; v = 2

si=0, ei=4; mi=(0+4)/2=2
A[ ] = {1, 2, 3, 4, 5}; A[2]=3(>2)
si=0, ei=mi-1=1; mi=0
A[ ] = {1, 2}, 3, 4, 5}; A[0]=1(<2)
si=mi+1=1, ei=1; mi=1
A[ ] = {1, {2}, 3, 4, 5}; A[1]=2

Ex-2: A[5] = {1, 2, 3, 4, 5}; v = 0

si=0, ei=4; mi=(0+4)/2=2
A[ ] = {1, 2, 3, 4, 5}; A[2]=3(>0)
si=0, ei=mi-1=1; mi=0
A[ ] = {1, 2}, 3, 4, 5}; A[0]=1(>0)
si=0, ei=mi-1=-1
A[ ] = {1, 2, 3, 4, 5}; not-found(-1)

Variants of Binary Search

Variant-1: [a/b partition]

$1/3 \sim 2/3$ partition – Break the array (of size n) into two parts (of size $\frac{n}{3}$ and $\frac{2n}{3}$), instead of breaking into two equal size halves.

- Choose the $\frac{1}{3}$-rd element $A[\frac{n}{3}]$
- If $v == A[\frac{n}{3}]$, we are done [1 comparison]
- If $v < A[\frac{n}{3}]$, search for v between $A[0]$ and $A[\frac{n}{3} - 1]$
- If $v > A[\frac{n}{3}]$, search for v between $A[\frac{n}{3} + 1]$ and $A[n - 1]$
- Repeat until v is found or no more elements remain for searching.

Variant-2: [d-ary partition]

3-ary $1/3$ partitions – Break the array (of size n) into three equal parts (of size $\frac{n}{3}$ each), instead of breaking into two equal size halves.

- Choose the $\frac{1}{3}$-rd and $\frac{2}{3}$-rd element $A[\frac{n}{3}]$ and $A[\frac{2n}{3}]$, respectively.
- If $(v == A[\frac{n}{3}])$ or $(v == A[\frac{2n}{3}])$, we are done [2 comparisons]
- If $v < A[\frac{n}{3}]$, search for v between $A[0]$ and $A[\frac{n}{3} - 1]$
- If $A[\frac{n}{3}] < v < A[\frac{2n}{3}]$, search for v between $A[\frac{n}{3} + 1]$ and $A[\frac{2n}{3}] - 1]$
- If $v > A[\frac{2n}{3}]$, search for v between $A[\frac{2n}{3} + 1]$ and $A[n - 1]$
- Repeat until v is found or no more elements remain for searching.

Number of Comparisons in Binary Search

Let, $C(n)$ = Max. no. of comparisons required for searching from n elements

Binary Search: [$1/2$ partition]

$$\begin{aligned} C(n) &= C(\frac{n}{2}) + 1 = C(\frac{n}{2^2}) + (1+1) = C(\frac{n}{2^3}) + (1+1+1) \\ &= \cdots = C(\frac{n}{2^k}) + k = 1 + k = 1 + \log_2 n \quad [w/o\ l.o.g., n = 2^k] \end{aligned}$$

Binary Search (Variant-1): [$1/3 \sim 2/3$ partition]

$$\begin{aligned} C(n) &= C(\frac{2n}{3}) + 1 = C(\frac{n}{(\frac{3}{2})^2}) + (1+1) = C(\frac{n}{(\frac{3}{2})^3}) + (1+1+1) \\ &= \cdots = C(\frac{n}{(\frac{3}{2})^k}) + k = 1 + k = 1 + \log_{\frac{3}{2}} n \quad [w/o\ l.o.g., n = (\frac{3}{2})^k] \end{aligned}$$

Binary Search (Variant-2): [3-ary $1/3$ partitions]

$$\begin{aligned} C(n) &= C(\frac{n}{3}) + 2 = C(\frac{n}{3^2}) + (2+2) = C(\frac{n}{3^3}) + (2+2+2) \\ &= \cdots = C(\frac{n}{3^k}) + 2k = 1 + 2k = 1 + 2\log_3 n \quad [w/o\ l.o.g., n = 3^k] \end{aligned}$$

Please Note, $\log_2 n \leq \log_{\frac{3}{2}} n \leq \log_{\frac{b}{a}} n$ & $\log_2 n \leq 2\log_3 n \leq (d-1)\log_d n$

Thank You