

Algorithms 2015: Searching and Sorting

(Based on [Manber 1989])

Yih-Kuen Tsay

1 Binary Search

Searching a Sorted Sequence

Problem 1. Let x_1, x_2, \dots, x_n be a sequence of real numbers such that $x_1 \leq x_2 \leq \dots \leq x_n$. 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 $x_i = z$.

Idea: cut the search space in half by asking only one question.

Binary Search

```
function Find ( $z, Left, Right$ ) : integer;  
begin  
  if  $Left = Right$  then  
    if  $X[Left] = z$  then  $Find := Left$   
    else  $Find := 0$   
  else  
     $Middle := \lceil \frac{Left+Right}{2} \rceil$ ;  
    if  $z < X[Middle]$  then  
       $Find := Find(z, Left, Middle - 1)$   
    else  
       $Find := Find(z, Middle, Right)$   
end
```

Binary Search (cont.)

```
Algorithm Binary_Search ( $X, n, z$ );  
begin  
   $Position := Find(z, 1, n)$ ;  
end
```

1.1 Cyclically Sorted Sequence

Searching a Cyclically Sorted Sequence

Problem 2. Given a cyclically sorted list, find the position of the minimal element in the list (we assume, for simplicity, that this position is unique).

- Example 1:

– 1 2 3 4 5 6 7 8
 – [5 6 7 0 1 2 3 4]

– The 4th is the minimal element.

- Example 2:

– 1 2 3 4 5 6 7 8
 – [0 1 2 3 4 5 6 7]

– The 1st is the minimal element.

- To cut the search space in half, what question should we ask?

Cyclic Binary Search

Algorithm Cyclic_Binary_Search (X, n);

begin

$Position := Cyclic_Find(1, n)$;

end

function Cyclic_Find ($Left, Right$) : integer;

begin

if $Left = Right$ **then** $Cyclic_Find := Left$

else

$Middle := \lfloor \frac{Left+Right}{2} \rfloor$;

if $X[Middle] < X[Right]$ **then**

$Cyclic_Find := Cyclic_Find(Left, Middle)$

else

$Cyclic_Find := Cyclic_Find(Middle + 1, Right)$

end

1.2 “Fixpoints”

“Fixpoints”

Problem 3. Given a sorted sequence of distinct integers a_1, a_2, \dots, a_n , determine whether there exists an index i such that $a_i = i$.

- Example 1:

– 1 2 3 4 5 6 7 8
 – [-1 1 2 4 5 6 8 9]

– $a_4 = 4$ (there are more ...).

- Example 2:

– 1 2 3 4 5 6 7 8
 – [-1 1 2 5 6 8 9 10]

– There is no i such that $a_i = i$.

- Again, can we cut the search space in half by asking only one question?

A Special Binary Search

```
function Special_Find (Left, Right) : integer;
begin
  if Left = Right then
    if A[Left] = Left then Special_Find := Left
    else Special_Find := 0
  else
    Middle := ⌊  $\frac{Left+Right}{2}$  ⌋;
    if A[Middle] < Middle then
      Special_Find := Special_Find(Middle + 1, Right)
    else
      Special_Find := Special_Find(Left, Middle)
  end
end
```

A Special Binary Search (cont.)

```
Algorithm Special_Binary_Search (A, n);
begin
  Position := Special_Find(1, n);
end
```

1.3 Stuttering Subsequence

Stuttering Subsequence

Problem 4. Given two sequences A and B , find the maximal value of i such that B^i is a subsequence of A .

- If $B = xyzza$, then $B^2 = xyzyzzzzax$, $B^3 = xxxyyzzzzzzaxx$, etc.
- B is a subsequence of A if we can embed B inside A in the same order but with possible holes.
- For example, $B^2 = xyzyzzzzax$ is a subsequence of $xxzzyyyyxzzzzzzaxx$.

2 Interpolation Search

Interpolation Search

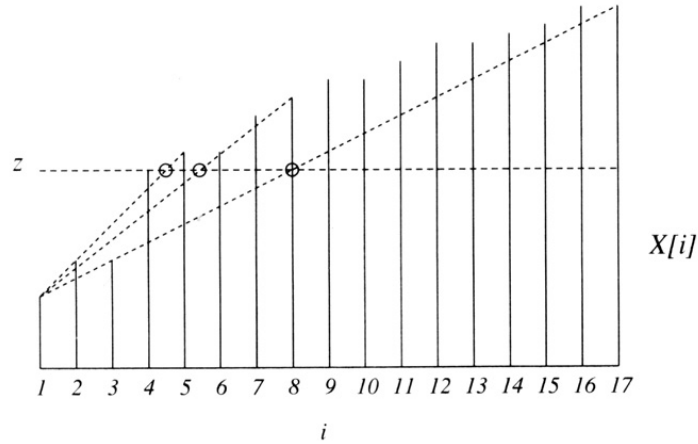


Figure 6.4 Interpolation search.

Source: [Manber 1989].

Interpolation Search (cont.)

```

function Int_Find (z, Left, Right) : integer;
begin
  if X[Left] = z then Int_Find := Left
  else if Left = Right or X[Left] = X[Right] then
    Int_Find := 0
  else
    Next_Guess := ⌈Left + ((z - X[Left]) * (Right - Left)) / (X[Right] - X[Left])⌉;
    if z < X[Next_Guess] then
      Int_Find := Int_Find(z, Left, Next_Guess - 1)
    else
      Int_Find := Int_Find(z, Next_Guess, Right)
  end
end

```

Interpolation Search (cont.)

```

Algorithm Interpolation_Search (X, n, z);
begin
  if z < X[1] or z > X[n] then Position := 0
  else Position := Int_Find(z, 1, n);
end

```

3 Sorting

Sorting

Problem 5. Given n numbers x_1, x_2, \dots, x_n , arrange them in increasing order. In other words, find a sequence of distinct indices $1 \leq i_1, i_2, \dots, i_n \leq n$, such that $x_{i_1} \leq x_{i_2} \leq \dots \leq x_{i_n}$.

A sorting algorithm is called **in-place** if no additional work space is used besides the initial array that holds the elements.

3.1 Using Balanced Search Trees

Using Balanced Search Trees

- Balanced search trees, such as AVL trees, may be used for sorting:
 1. Create an empty tree.
 2. Insert the numbers one by one to the tree.
 3. Traverse the tree and output the numbers.
- What's the time complexity? Suppose we use an AVL tree.

3.2 Radix Sort

Radix Sort

```
Algorithm Straight_Radix ( $X, n, k$ );  
begin  
  put all elements of  $X$  in a queue  $GQ$ ;  
  for  $i := 1$  to  $d$  do  
    initialize queue  $Q[i]$  to be empty  
  for  $i := k$  downto 1 do  
    while  $GQ$  is not empty do  
      pop  $x$  from  $GQ$ ;  
       $d :=$  the  $i$ -th digit of  $x$ ;  
      insert  $x$  into  $Q[d]$ ;  
    for  $t := 1$  to  $d$  do  
      insert  $Q[t]$  into  $GQ$ ;  
  for  $i := 1$  to  $n$  do  
    pop  $X[i]$  from  $GQ$   
end
```

3.3 Merge Sort

Merge Sort

```
Algorithm Mergesort ( $X, n$ );  
begin M_Sort(1,  $n$ ) end  
  
procedure M_Sort ( $Left, Right$ );  
begin  
  if  $Right - Left = 1$  then  
    if  $X[Left] > X[Right]$  then swap( $X[Left], X[Right]$ )  
  else if  $Left \neq Right$  then  
     $Middle := \lceil \frac{1}{2}(Left + Right) \rceil$ ;  
    M_Sort( $Left, Middle - 1$ );  
    M_Sort( $Middle, Right$ );  
end
```

Merge Sort (cont.)

```
 $i := Left$ ;  $j := Middle$ ;  $k := 0$ ;  
while ( $i \leq Middle - 1$ ) and ( $j \leq Right$ ) do  
   $k := k + 1$ ;
```

```

    if  $X[i] \leq X[j]$  then
         $TEMP[k] := X[i]; i := i + 1$ 
    else  $TEMP[k] := X[j]; j := j + 1;$ 
if  $j > Right$  then
    for  $t := 0$  to  $Middle - 1 - i$  do
         $X[Right - t] := X[Middle - 1 - t]$ 
for  $t := 0$  to  $k - 1$  do
     $X[Left + t] := TEMP[t]$ 
end

```

Merge Sort (cont.)

6	2	8	5	10	9	12	1	15	7	3	13	4	11	16	14	
②	⑥	8	5	10	9	12	1	15	7	3	13	4	11	16	14	
2	6	⑤	⑧	10	9	12	1	15	7	3	13	4	11	16	14	
②	⑤	⑥	⑧	10	9	12	1	15	7	3	13	4	11	16	14	
2	5	6	8	⑨	⑩	12	1	15	7	3	13	4	11	16	14	
2	5	6	8	9	10	①	⑫	15	7	3	13	4	11	16	14	
2	5	6	8	①	⑩	⑫	15	7	3	13	4	11	16	14	14	
①	②	⑤	⑥	⑧	⑩	⑩	⑫	15	7	3	13	4	11	16	14	
1	2	5	6	8	9	10	12	⑦	⑮	3	13	4	11	16	14	
1	2	5	6	8	9	10	12	7	15	③	⑬	⑮	4	11	16	14
1	2	5	6	8	9	10	12	③	⑦	⑬	⑮	⑮	4	11	16	14
1	2	5	6	8	9	10	12	3	7	13	15	④	⑪	⑮	16	14
1	2	5	6	8	9	10	12	3	7	13	15	4	11	⑭	⑯	⑰
1	2	5	6	8	9	10	12	③	④	⑦	⑪	⑬	⑭	⑮	⑯	⑰
①	②	③	④	⑤	⑥	⑦	⑧	⑨	⑩	⑪	⑫	⑬	⑭	⑮	⑯	⑰

Figure 6.8 An example of mergesort. The first row is in the initial order. Each row illustrates either an exchange operation or a merge. The numbers that are involved in the current operation are circled.

Source: [Manber 1989].

3.4 Quick Sort

Quick Sort

Algorithm Quicksort (X, n);

```

begin
     $Q\_Sort(1, n)$ 
end

```

procedure $Q_Sort(Left, Right)$;

```

begin
    if  $Left < Right$  then
         $Partition(X, Left, Right)$ ;
         $Q\_Sort(Left, Middle - 1)$ ;
         $Q\_Sort(Middle + 1, Right)$ 
    end
end

```

Quick Sort (cont.)

```

Algorithm Partition ( $X, Left, Right$ );
begin
     $pivot := X[Left]$ ;
     $L := Left$ ;  $R := Right$ ;
    while  $L < R$  do
        while  $X[L] \leq pivot$  and  $L \leq Right$  do  $L := L + 1$ ;
        while  $X[R] > pivot$  and  $R \geq Left$  do  $R := R - 1$ ;
        if  $L < R$  then  $swap(X[L], X[R])$ ;
     $Middle := R$ ;
     $swap(X[Left], X[Middle])$ 
end

```

Quick Sort (cont.)

6	2	8	5	10	9	12	1	15	7	3	13	4	11	16	14
6	2	4	5	10	9	12	1	15	7	3	13	8	11	16	14
6	2	4	5	3	9	12	1	15	7	10	13	8	11	16	14
6	2	4	5	3	1	12	9	15	7	10	13	8	11	16	14
1	2	4	5	3	6	12	9	15	7	10	13	8	11	16	14

Figure 6.10 Partition of an array around the pivot 6.

Source: [Manber 1989].

Quick Sort (cont.)

6	2	8	5	10	9	12	1	15	7	3	13	4	11	16	14
1	2	4	5	3	6	12	9	15	7	10	13	8	11	16	14
1	2	4	5	3	6	12	9	15	7	10	13	8	11	16	14
1	2	4	5	3	6	12	9	15	7	10	13	8	11	16	14
1	2	3	4	5	6	12	9	15	7	10	13	8	11	16	14
1	2	3	4	5	6	8	9	11	7	10	12	13	15	16	14
1	2	3	4	5	6	7	8	11	9	10	12	13	15	16	14
1	2	3	4	5	6	7	8	10	9	11	12	13	15	16	14
1	2	3	4	5	6	7	8	9	10	11	12	13	15	16	14
1	2	3	4	5	6	7	8	9	10	11	12	13	15	16	14
1	2	3	4	5	6	7	8	9	10	11	12	13	15	16	14
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Figure 6.12 An example of quicksort. The first line is the initial input. A new pivot is selected in each line. The pivots are circled. When a single number appears between two pivots it is obviously in the right position.

Source: [Manber 1989].

Average-Case Complexity of Quick Sort

- When $X[i]$ is selected (at random) as the pivot,

$$T(n) = n - 1 + T(i - 1) + T(n - i), \text{ where } n \geq 2.$$

The average running time will then be

$$\begin{aligned} T(n) &= n - 1 + \frac{1}{n} \sum_{i=1}^n (T(i - 1) + T(n - i)) \\ &= n - 1 + \frac{1}{n} \sum_{i=1}^n T(i - 1) + \frac{1}{n} \sum_{i=1}^n T(n - i) \\ &= n - 1 + \frac{1}{n} \sum_{j=0}^{n-1} T(j) + \frac{1}{n} \sum_{j=0}^{n-1} T(j) \\ &= n - 1 + \frac{2}{n} \sum_{i=0}^{n-1} T(i) \end{aligned}$$

- Solving this recurrence relation with full history, $T(n) = O(n \log n)$.

3.5 Heap Sort

Heap Sort

```

Algorithm Heapsort ( $A, n$ );
begin
    Build_Heap( $A$ );
    for  $i := n$  downto 2 do
        swap( $A[1], A[i]$ );
        Rearrange_Heap( $i - 1$ )
    end

```

Heap Sort (cont.)

```

procedure Rearrange_Heap ( $k$ );
begin
     $parent := 1$ ;
     $child := 2$ ;
    while  $child \leq k - 1$  do
        if  $A[child] < A[child + 1]$  then
             $child := child + 1$ ;
        if  $A[child] > A[parent]$  then
            swap( $A[parent], A[child]$ );
             $parent := child$ ;
             $child := 2 * child$ 
        else  $child := k$ 
    end

```

Heap Sort (cont.)

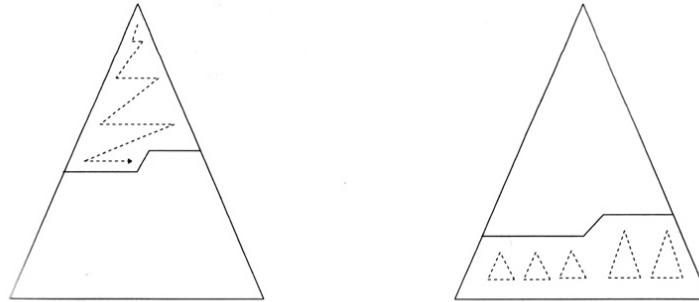


Figure 6.14 Top down and bottom up heap construction.

Source: [Manber 1989].

Building a Heap Bottom Up

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
6	2	8	5	10	9	12	1	15	7	3	13	4	11	16	14
2	6	8	5	10	9	12	14	15	7	3	13	4	11	16	1
2	6	8	5	10	9	16	14	15	7	3	13	4	11	12	1
2	6	8	5	10	13	16	14	15	7	3	9	4	11	12	1
2	6	8	15	10	13	16	14	5	7	3	9	4	11	12	1
2	6	16	15	10	13	12	14	5	7	3	9	4	11	8	1
2	15	16	14	10	13	12	6	5	7	3	9	4	11	8	1
16	15	13	14	10	9	12	6	5	7	3	2	4	11	8	1

Figure 6.15 An example of building a heap bottom up. The numbers on top are the indices. The circled numbers are those that have been exchanged on that step.

Source: [Manber 1989] (6 and 2 in the first row should be swapped).

A Lower Bound for Sorting

- A lower bound for a particular problem is a proof that *no algorithm* can solve the problem better.
- We typically define a computation model and consider only those algorithms that fit in the model.
- **Decision trees** model computations performed by *comparison-based* algorithms.

Theorem 6 (Theorem 6.1). *Every decision-tree algorithm for sorting has height $\Omega(n \log n)$.*

4 Order Statistics

Order Statistics: Minimum and Maximum

Problem 7. *Find the maximum and minimum elements in a given sequence.*

- The obvious solution requires $(n - 1) + (n - 2) (= 2n - 3)$ comparisons between elements.
- Can we do better? Which comparisons could have been avoided?

Order Statistics: *K*th-Smallest

Problem 8. Given a sequence $S = x_1, x_2, \dots, x_n$ of elements, and an integer k such that $1 \leq k \leq n$, find the k th-smallest element in S .

Order Statistics: *K*th-Smallest (cont.)

```
procedure Select (Left, Right, k);
begin
  if Left = Right then
    Select := Left
  else Partition(X, Left, Right);
    let Middle be the output of Partition;
    if Middle - Left + 1  $\geq$  k then
      Select(Left, Middle, k)
    else
      Select(Middle + 1, Right, k - (Middle - Left + 1))
end
```

Order Statistics: *K*th-Smallest (cont.)

The nested “if” statement may be simplified:

```
procedure Select (Left, Right, k);
begin
  if Left = Right then
    Select := Left
  else Partition(X, Left, Right);
    let Middle be the output of Partition;
    if Middle  $\geq$  k then
      Select(Left, Middle, k)
    else
      Select(Middle + 1, Right, k)
end
```

Order Statistics: *K*th-Smallest (cont.)

```
Algorithm Selection (X, n, k);
begin
  if (k < 1) or (k > n) then print “error”
  else S := Select(1, n, k)
end
```

5 Finding a Majority

Finding a Majority

Problem 9. Given a sequence of numbers, find the majority in the sequence or determine that none exists.

A number is a *majority* in a sequence if it occurs more than $\frac{n}{2}$ times in the sequence.

Finding a Majority (cont.)

Algorithm Majority (X, n);

begin

$C := X[1]; M := 1;$

for $i := 2$ **to** n **do**

if $M = 0$ **then**

$C := X[i]; M := 1$

else

if $C = X[i]$ **then** $M := M + 1$

else $M := M - 1;$

Finding a Majority (cont.)

if $M = 0$ **then** $Majority := -1$

else

$Count := 0;$

for $i := 1$ **to** n **do**

if $X[i] = C$ **then** $Count := Count + 1;$

if $Count > n/2$ **then** $Majority := C$

else $Majority := -1$

end