Searching Techniques

- Information retrieval is one of the most important applications of computer
- Given a name, retrieve associated information
- Dictionary/directory look-up
Types of Searching

- **Sequential Searching**
  - Can search from unordered list
  - Time complexity $\Rightarrow O(n)$

- **Binary searching**
  - Can search from ordered list only
  - Time complexity $\Rightarrow O(\log_2 n)$

- **Ternary searching**
  - Can search from ordered list only
  - Time complexity $\Rightarrow O(\log_3 n)$

- **Index searching**
  - Can search from ordered list only
  - Uses index table
Sequential Search Algorithm

Input n, x(i), i=1,...,n; key
found = false;
i=1
while i ≤ n and not (found)
{
    if key = x(i) then found = true
    else i = i+1
}

If found then
    print “key is found”
else
    print “key is not found”

Complexity \(\Theta(n)\)
Binary Search Algorithm

Input n, x(i), i=1,…,n; key found = false;
L=1, U=n;
While L ≤ U and not(found)
{ mid = (L+U)/2;
  if key = x(mid) then found = true
  else if key < x(mid) then U=mid-1
  else L=mid+1
}
If found then
  print “key is found”
else
  print “key is not found”

Complexity \( \Rightarrow O(\log_2 n) \)
Ternary Search Algorithm

- Divide list into three equal portions
- \([1, \, n/3] ; [n/3 +1, 2n/3] \) and \([2n/3 +1, n]\)
- Find out the range in which the key belongs and search that range using ternary search.

Complexity \(\rightarrow O(\log_3 n)\)
Indexed Search

Index Table

List
Indexed Search – Contd…

```
2 16 29

2 10 16 23 29 37

2 4 5 7 10 12 14 16 18 20 23 26 29 32 35 37 38 40
```

List

Index
Sorting Algorithms

- Sorting is an important concept in real life.
- Always better to keep things in some sensible order.
- Useful for searching
- Various Techniques for sorting
  - Selection sort
  - Insertion sort
  - Bubble sort
Selection Sort

- Algorithm for arranging elements of list in ascending order.
  - In unordered list choose minimum and swap with first element of unsorted list.
  - Reduce the list size by one
  - Repeat the process till list reduces to 0 size.
Selection Sort - contd

Continue
Selection Sort - Algorithm

Input $n$, $x(i)$, $i=1,\ldots,n$

$i=1$

while $i \leq n$

{ 

  - Find min of list $[x(i), \ldots x(n)]$
    
    - let min = $x(j)$

    {# position of min}
    
    - Swap $x(i)$ with $x(j)$

    - $i=i+1$

}

print $x(i)$, $i=1,\ldots,n$
Minimum - Algorithm

Statements for finding minimum of list \( x(i), \ldots, x(n) \)

```plaintext
min = x(i);
j = i

# j is index at which min resides
k = i + 1
while k <= n
    if x(k) < min then
        min = x(k); j = k
    k = k + 1
```

Swap - statements

- Swap the values of $x(i)$ and $x(j)$. Here
  - $x(i)$ is the first element of unsorted list
  - $x(j)$ is minimum element

\[
\begin{align*}
  t &= x(i) \\
  x(i) &= x(j) \\
  x(j) &= t
\end{align*}
\]
Complete Selection Sort Algorithm

```plaintext
input n, x(i), i=1,…,n
i=1
while i ≤ n
    { min = x(i); j=i ; k=i+1
        while k ≤ n
            { if x(k) < min
                then { min = x(k); j=k }
                k=k+1
            }
    } t = x(i); x(i) = x(j); x(j) = t
    i=i+1
} print x(i), i=1,…,n
```

Time complexity → O(n²)
Space complexity → O(1)
# Input elements of a list and terminate by say, 1000

def read_list(s):
    x = input('Input the value and terminate by 1000
    while x != 1000:
        s = s + [x]
        x = input()
    return s

s = []
s = read_list(s)
print '\nlist elements are: \n', s

>>> 'Input the value and terminate by 1000
5
4
8
1000
>>> [5, 4, 8]
def selection_sort(s):
    i=0
    n=len(s)
    while i < n:
        k = min(s,i,n)
        # swap s[i] with s[k]
        t=s[i] ; s[i] = s[k]; s[k]=t
        i=i+1
    print 'List at each iteration
', s
    return s

s=[]
s=read_list(s)
s=selection_sort(s)
print 'sorted list
', s

>>> s= [8,7,3,4]
List at each iteration
[3,7,8,4]
List at each iteration
[3,4,8,7]
List at each iteration
[3,4,7,8]
Insertion Sort

- At each iteration
  - From unordered list choose the first element and insert it in the sorted.
  - Reduce the list size by one
  - Repeat the process till list reduces to 0 size.
Selection Sort - contd
Insertion Sort - Algorithm

input n, x(i), i=1,…,n
i=2
while i ≤ n
    { 
    - insert x(i) in the sorted list
      x(1),…, x(i-1)
    - i=i+1
    }
print x(i), i=1,…,n
Insert - Algorithm

Statements for inserting $x(i)$ in the list $x(1), \ldots x(i-1)$

```plaintext
k = i-1;
val = x(i)
while val < x(k)
    { x(k+1) = x(k)
        k = k-1
    }

x(k+1) = val
```
Complete Insertion Sort Algorithm

Input n, x(i), i=1,…,n
i=2
While i ≤ n
    { k = i-1; val = x(i)
        While val < x(k)
            { x(k+1) = x(k)
                k=k-1
            }
        x(k+1) = val
        i=i+1
    }
print x(i), i=1,…,n

Time complexity \( \Theta(n^2) \)
Space Complexity \( \Theta(1) \)
def insertion(s):
    k=1; n=len(s)
    while k < n:
        insert(s,s[k],0,k-1)
        k=k+1
    return s

s=[]
s=read_list(s);
s=insertion(s)
print '\nsorted list\n',s

def insert(s,x,low, up):
    j=up; i=low
    while x < s[j] and j>=i:
        s[j+1] = s[j]; j=j-1
    s[j+1]=x
    return s

>>> s=[5,4,1,2]
List at each iteration
[4,5,1,2]
List at each iteration
[1,4,5,2]
List at each iteration
[1,2,4,5]
Bubble Sort

- In each iteration, exchange adjacent elements if not in proper order
- The highest element gets settled at the bottom of the list.
- Each time list is reduced by one from start to end-1)
- Stop the process
  - If no exchange is done OR
  - the list reduces to 1 size.
## Bubble Sort - contd

<table>
<thead>
<tr>
<th>8</th>
<th>5</th>
<th>9</th>
<th>1</th>
<th>7</th>
<th>3</th>
<th>2</th>
<th>4</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>8</td>
<td>1</td>
<td>7</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>7</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

No exchange, so list is sorted and exit
Bubble Sort - Algorithm

input n, x(i), i=1,...,n
found=false;
while not(found)
    { - exchange pair of elements, if
        not in order
        - If no exchange then found =true
    }
print x(i), i=1,...,n
Loop for exchanging pair if required - Algorithm

\[ j = 1 \; ; \; \text{End}_{\text{elt}} = M \]
exchange = false
while \( j < \text{End}_{\text{elt}} \)
{  if \( x(j) > x(j+1) \) then
{    \( t = x(j) \);
    \( x(j) = x(j+1) \);
    \( x(j+1) = t \)
    exchange = true
  }
  j = j + 1
}

Bubble Sort - Algorithm

input n, x(i), i=1,...,n
found=false; i=1
while not (found)
{ j = 1; exchange = false; End_elt = n-i+1
  while j < End_elt
  { if x(j) > x(j+1) then
    { t=x(j); x(j) = x(j+1); x(j+1)= t
      exchange=true
    }
    j=j+1
  }
  if exchange = false then found true
  i=i+1
}
}; print x(i), i=1,...,n
def bub(s):
    found=0 ; i=0;  n=len(s)
    while found !=1:
        j = 0; exchange = 0
        while j < n-i-1:
            if s[j] > s[j+1]:
                t=s[j];    s[j] = s[j+1];    s[j+1] = t
                exchange=1
            j=j+1
        if exchange == 0: found = 1
        i=i+1
    return s

s=[];  s=read_list(s);  s=bub(s)
print '\nsorted list\n', s

>>> s= [5,4,8,2,1]
List at each iteration
[4,5,2,1,8]
List at each iteration
[4,2,1,5,8]
List at each iteration
[2,1,4,5,8]
List at each iteration
[1,2,4,5,8]
Partitioning of a List

Partitioning a given unordered list into two subsets such that

- all elements in one set $\leq x$ and
- other elements in other list are $> x$

Applications of Partitioning

- Sorting
- Median finding
- Statistical classification
Basic steps – Partition around x

- Initially set i to start index of the list and j to be the last index of the list
- Move towards middle first from left and then from right as follows:
  - Move i from left to right (increment i) till we encounter a number > x
  - Then move j from right to left (decrement j) till we get a number ≤ x
- Exchange wrongly partitioned pair
- Start the process from the existing positions till i crosses j.
**Example → x=11**

<table>
<thead>
<tr>
<th>10</th>
<th>20</th>
<th>8</th>
<th>25</th>
<th>6</th>
<th>35</th>
<th>14</th>
<th>26</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>20</td>
<td>8</td>
<td>25</td>
<td>6</td>
<td>35</td>
<td>14</td>
<td>26</td>
</tr>
</tbody>
</table>

Initially i=1; j=9

Increment i till we get a number > x

<table>
<thead>
<tr>
<th>10</th>
<th>20</th>
<th>8</th>
<th>25</th>
<th>6</th>
<th>35</th>
<th>14</th>
<th>26</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>20</td>
<td>8</td>
<td>25</td>
<td>6</td>
<td>35</td>
<td>14</td>
<td>26</td>
</tr>
</tbody>
</table>

Now decrement j till we get a number < x

<table>
<thead>
<tr>
<th>10</th>
<th>20</th>
<th>8</th>
<th>25</th>
<th>6</th>
<th>35</th>
<th>14</th>
<th>26</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>20</td>
<td>8</td>
<td>25</td>
<td>6</td>
<td>35</td>
<td>14</td>
<td>26</td>
</tr>
</tbody>
</table>

Now i=2; j=5
Example Contd…

Exchange i and j position values

Move i forward till i > j

Since i<j, so terminate the process
Partitioning Algorithm

Input list \( L(i), i=1, n \)
\( i=1; j=n \)
While (\( i \leq j \))
\{ While (\( L(i) < x \))
\{ i=i+1 \}
\}
While (\( L(j) > x \))
\{ j=j-1 \}
Swap \( L(i) \) and \( L(j) \) values
\( i=i+1 \)
\( j=j-1 \)
\}
Output \( L(k), k=1,j \) and \( L(k), k=i \) to \( n \)