Math 2603 - Lecture 12
Section 8.3 Searching and sorting

Bo Lin

October 1st, 2019

Searching

Motivation

When solving problem, the following **searching** tasks are very common.

- Given numbers $a_1, \ldots, a_n$, check whether a particular number x appears among them. (Example: set membership)
- Given sorted real numbers $a_1 \leq a_2 \leq \ldots \leq a_n$ and another real number x, find an index i such that $a_i \leq x < a_{i+1}$. (Example: floor and ceiling functions)

Motivation

When solving problem, the following **searching** tasks are very common.

- Given numbers $a_1, \ldots, a_n$, check whether a particular number x appears among them. (Example: set membership)
- Given sorted real numbers $a_1 \leq a_2 \leq \ldots \leq a_n$ and another real number x, find an index i such that $a_i \leq x < a_{i+1}$. (Example: floor and ceiling functions)

They are required by many other processes and are repeated many times, so we would like to find efficient algorithms for them.

A straightforward approach

For the first type of searching task, a straightforward approach would be: compare x with every a_i.

A straightforward approach

For the first type of searching task, a straightforward approach would be: compare x with every a_i.
In general, if the data $a_1, \ldots, a_n$ do not have any pattern, there is no shortcut. The reason is very simple: every a_i could be x, so in the worst case when x is not equal to all other a_j for $j \neq i$, one still cannot ignore a_i.

A linear search algorithm

input : Real numbers $a_1, \ldots, a_n$ and x.
output: If x appear among a_i's, "True"; otherwise, "False".

for $i \leftarrow 1$ **to** n **do**
    if $x = a_i$ **then**
        output "True";
        set $i = 2n$;
    end
end
Output "False;

Algorithm 1: Linear Search

Example: linear search

Example

Suppose $(a_1, a_2, a_3, a_4) = (6, 0, -2, 1)$ *and* $x = -2$.

Example: linear search

Example

Suppose $(a_1, a_2, a_3, a_4) = (6, 0, -2, 1)$ *and* $x = -2$.

Solution

- $i = 1$: *compare* x *with* $a_1 = 6$, *not equal;*

Example: linear search

Example

Suppose $(a_1, a_2, a_3, a_4) = (6, 0, -2, 1)$ *and* $x = -2$.

Solution

- $i = 1$: *compare* x *with* $a_1 = 6$, *not equal;*
- $i = 2$: *compare* x *with* $a_2 = 0$, *not equal;*

Example: linear search

Example

Suppose $(a_1, a_2, a_3, a_4) = (6, 0, -2, 1)$ *and* $x = -2$.

Solution

- $i = 1$: *compare* x *with* $a_1 = 6$, *not equal;*
- $i = 2$: *compare* x *with* $a_2 = 0$, *not equal;*
- $i = 3$: *compare* x *with* $a_3 = -2$, **equal**;

Example: linear search

Example

Suppose $(a_1, a_2, a_3, a_4) = (6, 0, -2, 1)$ *and* $x = -2$.

Solution

- $i = 1$: *compare* x *with* $a_1 = 6$, *not equal;*
- $i = 2$: *compare* x *with* $a_2 = 0$, *not equal;*
- $i = 3$: *compare* x *with* $a_3 = -2$, **equal**;
- *output "True";*

Example: linear search

Example

Suppose $(a_1, a_2, a_3, a_4) = (6, 0, -2, 1)$ *and* $x = -2$.

Solution

- $i = 1$: *compare* x *with* $a_1 = 6$, *not equal;*
- $i = 2$: *compare* x *with* $a_2 = 0$, *not equal;*
- $i = 3$: *compare* x *with* $a_3 = -2$, **equal**;
- *output "True";*
- *set* $i = 2n = 8$;

Example: linear search

Example

Suppose $(a_1, a_2, a_3, a_4) = (6, 0, -2, 1)$ *and* $x = -2$.

Solution

- $i = 1$: *compare* x *with* $a_1 = 6$, *not equal;*
- $i = 2$: *compare* x *with* $a_2 = 0$, *not equal;*
- $i = 3$: *compare* x *with* $a_3 = -2$, **equal**;
- *output "True";*
- *set* $i = 2n = 8$;
- *since* $i = 8 > 4 = n$, *the algorithm terminates.*

An analysis

Remark

The purpose of setting $i = 2n$ is to indicate that we already found x in the search. And i has a value not in 1 to n, so the loop stops immediately.

An analysis

Remark

The purpose of setting $i = 2n$ is to indicate that we already found x in the search. And i has a value not in 1 to n, so the loop stops immediately.

Example

What is the complexity function of this algorithm?

An analysis

Remark

The purpose of setting $i = 2n$ is to indicate that we already found x in the search. And i has a value not in 1 to n, so the loop stops immediately.

Example

What is the complexity function of this algorithm? For each i, we need one comparison between x and a_i. Is it all it takes?

An analysis

Remark

The purpose of setting $i = 2n$ is to indicate that we already found x in the search. And i has a value not in 1 to n, so the loop stops immediately.

Example

What is the complexity function of this algorithm? For each i, we need one comparison between x and a_i. Is it all it takes?
Please note that, we still need to check whether the procedure is over, which means whether $i = n$. So it takes 2 comparisons for each i, and the complexity function would be $2n = \mathcal{O}(n)$.

An analysis

Remark

The purpose of setting $i = 2n$ is to indicate that we already found x in the search. And i has a value not in 1 to n, so the loop stops immediately.

Example

What is the complexity function of this algorithm? For each i, we need one comparison between x and a_i. Is it all it takes?
Please note that, we still need to check whether the procedure is over, which means whether $i = n$. So it takes 2 comparisons for each i, and the complexity function would be $2n = \mathcal{O}(n)$.

Remark

If the data is well-organized, we have a more efficient searching algorithm.

Binary search

Example

Suppose I have a roster of Math 2603, where your first names are listed in alphabetical order. Now I am looking for a student with first name Tony, how should I do?

Binary search

Example

Suppose I have a roster of Math 2603, where your first names are listed in alphabetical order. Now I am looking for a student with first name Tony, how should I do?
Looking from the top may not be a good idea, because the initial T *is the* 20*th letter among the* 26 *letters in the English alphabet. So I probably focus on the second half of the roster.*

Binary search

Example

Suppose I have a roster of Math 2603, where your first names are listed in alphabetical order. Now I am looking for a student with first name Tony, how should I do?
Looking from the top may not be a good idea, because the initial T is the 20th letter among the 26 letters in the English alphabet. So I probably focus on the second half of the roster.

Remark

This is exactly the motivation of **binary search**. *It's advantage is that each time we can drop half of the data and narrow down the space we need to search next.*

A binary search algorithm

input : Real numbers $a_1 \leq a_2 \leq \cdots \leq a_n$ and x.
output: If x appear among a_i's, "True"; otherwise, "False".
while $n > 0$ **do**
    if $n = 1$ **then**
        if $x = a_1$ **then** output "True"; set $n = 0$;
        else output "False"; set $n = 0$;
    end
    else
        Set $m = \lfloor \frac{n}{2} \rfloor$;
        if $x = a_m$ **then** output "True"; set $n = 0$;
        else if $x < a_m$ **then** replace the current list with $a_1, \cdots, a_m$; set $n = m$;
        else replace the current list with $a_{m+1}, \cdots, a_n$; set $n = n - m$;
    end
end

Algorithm 2: Binary Search

Example: binary search

Example

Suppose $x = 12$ *and*

a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}
5	6	7	10	11	12	15	17	19	20

Example: binary search

Example

Suppose $x = 12$ *and*

a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}
5	6	7	10	11	12	15	17	19	20

Solution

- $n = 10 > 0$; $m = \lfloor \frac{10}{2} \rfloor = 5$.

Example: binary search

Example

Suppose $x = 12$ *and*

a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}
5	6	7	10	11	12	15	17	19	20

Solution

- $n = 10 > 0$; $m = \lfloor \frac{10}{2} \rfloor = 5$.
- *compare* $x = 12$ *with* $a_m = a_5 = 11$, $x \leq a_m$ *is false.*

Example: binary search

Example

Suppose $x = 12$ *and*

a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}
5	6	7	10	11	12	15	17	19	20

Solution

- $n = 10 > 0$; $m = \lfloor \frac{10}{2} \rfloor = 5$.
- *compare* $x = 12$ *with* $a_m = a_5 = 11$, $x \leq a_m$ *is false.*
- *replace the list by* $(a_1, a_2, a_3, a_4, a_5) = (12, 15, 17, 19, 20)$; *replace* n *by* $n - m = 10 - 5$.

Example: binary search

Example

Suppose $x = 12$ *and*

a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}
5	6	7	10	11	12	15	17	19	20

Solution

- $n = 10 > 0$; $m = \lfloor \frac{10}{2} \rfloor = 5$.
- *compare* $x = 12$ *with* $a_m = a_5 = 11$, $x \leq a_m$ *is false.*
- *replace the list by* $(a_1, a_2, a_3, a_4, a_5) = (12, 15, 17, 19, 20)$; *replace* n *by* $n - m = 10 - 5$.
- $n = 5 > 0$; $m = \lfloor \frac{5}{2} \rfloor = 2$.

Example: binary search

Example

Suppose $x = 12$ *and*

a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}
5	6	7	10	11	12	15	17	19	20

Solution

- $n = 10 > 0$; $m = \lfloor \frac{10}{2} \rfloor = 5$.
- *compare* $x = 12$ *with* $a_m = a_5 = 11$, $x \le a_m$ *is false.*
- *replace the list by* $(a_1, a_2, a_3, a_4, a_5) = (12, 15, 17, 19, 20)$; *replace* n *by* $n - m = 10 - 5$.
- $n = 5 > 0$; $m = \lfloor \frac{5}{2} \rfloor = 2$.
- *compare* $x = 12$ *with* $a_m = a_2 = 15$, $x \le a_m$ *is true.*

Example: binary search

Example

Suppose $x = 12$ *and*

a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}
5	6	7	10	11	12	15	17	19	20

Solution

- $n = 10 > 0$; $m = \lfloor \frac{10}{2} \rfloor = 5$.
- *compare* $x = 12$ *with* $a_m = a_5 = 11$, $x \leq a_m$ *is false.*
- *replace the list by* $(a_1, a_2, a_3, a_4, a_5) = (12, 15, 17, 19, 20)$; *replace* n *by* $n - m = 10 - 5$.
- $n = 5 > 0$; $m = \lfloor \frac{5}{2} \rfloor = 2$.
- *compare* $x = 12$ *with* $a_m = a_2 = 15$, $x \leq a_m$ *is true.*
- *replace the list by* $(a_1, a_2) = (12, 15)$; *replace* n *by* $m = 2$.

Example: binary search

Solution

- $n = 2 > 0$; $m = \lfloor \frac{2}{2} \rfloor = 1$.
- *compare* $x = 12$ *with* $a_m = a_1 = 12$, $x \leq a_m$ *is true.*
- *output "True"; Set* $n = 0$.

Example: binary search

Solution

- $n = 2 > 0$; $m = \lfloor \frac{2}{2} \rfloor = 1$.
- *compare* $x = 12$ *with* $a_m = a_1 = 12$, $x \leq a_m$ *is true.*
- *output "True"; Set* $n = 0$.
- $n = 0$; *the algorithm terminates.*

Complexity analysis

We first assume that $n = 2^k$ for some $k \in \mathbb{N}$. Then there are at most k rounds of list replacement.

Complexity analysis

We first assume that $n = 2^k$ for some $k \in \mathbb{N}$. Then there are at most k rounds of list replacement. In each round, the comparison steps are: $n > 0$? $n = 1$? $x \le a_m$? So 3 comparisons. Finally there is one more comparison when $n = 1$: $x = a_1$?

Remark

If $n = 2^k$, the complexity of the binary search algorithm is $3k + 1 = 3\log_2 n$. For general n, the order remains the same, so this algorithm has complexity $\mathcal{O}(\log n)$.

Sorting

Motivation

Remark

Comparing the two algorithms of searching, we can see that a sorted pattern of the data is very useful to simplify other operations.

Motivation

Remark

Comparing the two algorithms of searching, we can see that a sorted pattern of the data is very useful to simplify other operations.

Definition

A **sorting algorithm** *puts a list of numbers into increasing order or a list of words into alphabetical order.*

Bubble Sort

Remark

Suppose we want to sort a list of numbers $a_1, \cdots, a_n$ *in increasing order. Then which number should be in the end?*

Bubble Sort

Remark

Suppose we want to sort a list of numbers $a_1, \cdots, a_n$ *in increasing order. Then which number should be in the end? The largest number among the list.*

Bubble Sort

Remark

Suppose we want to sort a list of numbers $a_1, \cdots, a_n$ in increasing order. Then which number should be in the end? The largest number among the list. So we can go through the list and try to pass the largest number all the way to the right. This is the motivation of the **bubble sort**.

Bubble Sort

Remark

Suppose we want to sort a list of numbers $a_1, \cdots, a_n$ in increasing order. Then which number should be in the end? The largest number among the list. So we can go through the list and try to pass the largest number all the way to the right. This is the motivation of the **bubble sort**.

In the first round, we compare a_1 and a_2 and then put the larger one as new a_2;

Bubble Sort

Remark

Suppose we want to sort a list of numbers $a_1, \cdots, a_n$ in increasing order. Then which number should be in the end? The largest number among the list. So we can go through the list and try to pass the largest number all the way to the right. This is the motivation of the **bubble sort**.

In the first round, we compare a_1 and a_2 and then put the larger one as new a_2; next we compare a_2 and a_3, then put the larger one as new a_3; and so on ...

Remark

One largest number in the list will reach a_n after this round. Then we repeat the process for the remaining $n-1$ numbers.

Bubble Sort algorithm

input : Real numbers $a_1, a_2, \cdots, a_n$
output: The same list of numbers in increasing order
for $i = n-1$ *down* **to** 1 **do**
    for $j = 1$ **to** i **do**
        if $a_j > a_{j+1}$ **then**
            swap a_j and a_{j+1}.
        end
    end
end
output $a_1, a_2, \cdots, a_n$.

Algorithm 3: Bubble sort

Bubble Sort algorithm

input : Real numbers $a_1, a_2, \cdots, a_n$
output: The same list of numbers in increasing order
for $i = n - 1$ *down* **to** 1 **do**
    for $j = 1$ **to** i **do**
        if $a_j > a_{j+1}$ **then**
            swap a_j and a_{j+1}.
        end
    end
end
output $a_1, a_2, \cdots, a_n$.

Algorithm 4: Bubble sort

Remark

Complexity function is $\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} = \mathcal{O}(n^2)$.

Example: bubble sort

Example

Apply bubble sort to the list $3, 1, 7, 2, 5, 4$.

Example: bubble sort

Example

Apply bubble sort to the list $3, 1, 7, 2, 5, 4$.

Solution

- *round* 1*:* $\mathbf{31}7254 \to 1\mathbf{37}254 \to 13\mathbf{72}54 \to 132\mathbf{75}4 \to 1325\mathbf{74} \to 132547$.

Example: bubble sort

Example

Apply bubble sort to the list $3, 1, 7, 2, 5, 4$.

Solution

- *round* 1: $\mathbf{31}7254 \to 1\mathbf{37}254 \to 13\mathbf{72}54 \to 132\mathbf{75}4 \to 1325\mathbf{74} \to 132547$.
- *round* 2: $\mathbf{13}2547 \to 1\mathbf{32}547 \to 12\mathbf{35}47 \to 123\mathbf{54}7 \to 123457$.

Example: bubble sort

Example

Apply bubble sort to the list $3, 1, 7, 2, 5, 4$.

Solution

- *round* 1: $\mathbf{31}7254 \to 1\mathbf{37}254 \to 13\mathbf{72}54 \to 132\mathbf{75}4 \to 1325\mathbf{74} \to 132547$.
- *round* 2: $\mathbf{13}2547 \to 1\mathbf{32}547 \to 12\mathbf{35}47 \to 123\mathbf{54}7 \to 123457$.
- *round* 3: *no more swap needed; output* $1, 2, 3, 4, 5, 7$ *and terminate.*

Merging

Remark

There are sorting algorithms with complexity better than $\mathcal{O}(n^2)$.

Merging

Remark

There are sorting algorithms with complexity better than $\mathcal{O}(n^2)$.

Example

Suppose for 2 *studio sections of Math 2603, I have sorted your exam papers in alphabetical order. How can I combine these papers in alphabetical order?*

Merging

Remark

There are sorting algorithms with complexity better than $\mathcal{O}(n^2)$.

Example

Suppose for 2 studio sections of Math 2603, I have sorted your exam papers in alphabetical order. How can I combine these papers in alphabetical order?

Solution

First I compare the first paper in each section. Suppose they are Alice from section A and Bob from section B. Then Alice would be the very first name after combination. Now who could be the overall 2nd?

Merging

Remark

There are sorting algorithms with complexity better than $\mathcal{O}(n^2)$.

Example

Suppose for 2 studio sections of Math 2603, I have sorted your exam papers in alphabetical order. How can I combine these papers in alphabetical order?

Solution

First I compare the first paper in each section. Suppose they are Alice from section A and Bob from section B. Then Alice would be the very first name after combination. Now who could be the overall 2nd? If from section B, it must be Bob; if from section A, it must be the 2nd name in section A. So it turns out that in every step, I need to do a single comparison.

Merging algorithm

input : Sorted lists $L_1 : a_1 \leq a_2 \leq \cdots \leq a_s$ and
$L_2 : b_1 \leq b_2 \leq \cdots \leq b_t$
output: The union of L_1 and L_2 as a sorted list
$L_3 : c_1 \leq c_2 \leq \cdots \leq c_{s+t}$
set L_3 to be an empty list; set $i = 1$; set $j = 1$;
while $i \leq s$ & $j \leq t$ **do**
  if $a_i > b_j$ **then** append b_j to the end of L_3; set $j = j + 1$;
  else append a_i to the end of L_3; set $i = i + 1$;
end
if $i > s$ **then** append $b_j, \cdots, b_t$ to the end of L_3 ;
else if $j > t$ **then** append $a_i, \cdots, a_s$ to the end of L_3 ;
output L_3.

Algorithm 5: Merging

Merge sort

Remark

The complexity of the above merging algorithm is $s + t - 1$*, which enables the* **merge sort** *algorithm.*

Merge sort

Remark

The complexity of the above merging algorithm is $s + t - 1$*, which enables the* **merge sort** *algorithm.*

The idea is simple: given n numbers, we divide them into two parts, and we try to sort both parts first, then merge the two parts. When sorting each part, we apply the same approach.

Merge sort

Remark

The complexity of the above merging algorithm is $s + t - 1$, which enables the **merge sort** *algorithm.*

The idea is simple: given n numbers, we divide them into two parts, and we try to sort both parts first, then merge the two parts. When sorting each part, we apply the same approach. So essentially it is a recursive algorithm.

Merge sort algorithm

```
input  : Real numbers a_1, a_2, ..., a_n
output: The same list of numbers in increasing order
for i ← 1 to n do
    set list L_i be the single element a_i;
end
set F = 0;
while F = 0 do
    if n = 1 then  set F = 1; output L_1 ;
    else if n = 2m is even then  for i ← 1 to m do
        merge sorted lists L_{2i−1} and L_{2i} and sort; label the new list L_i;
    end
    set n = m ;
    else if n = 2m + 1 is odd then  for i ← 1 to m do
        merge sorted lists L_{2i−1} and L_{2i} and sort; label the new list L_i;
        set L_{m+1} = L_n;
    end
    set n = m + 1 ;
end
```

Algorithm 6: Merge sort

Example: merge sort

Example

Sort $2, 9, 1, 4, 6, 5, 3.$

Example: merge sort

Example

Sort $2, 9, 1, 4, 6, 5, 3$.

Solution

- *Round* 1*:* $n = 7, m = 3$*;* $L_i : a_i$ *for* $1 \leq i \leq 7$*.*

Example: merge sort

Example

Sort $2, 9, 1, 4, 6, 5, 3$.

Solution

- *Round* 1: $n = 7, m = 3$; $L_i : a_i$ *for* $1 \leq i \leq 7$.
- *Round* 2: $n = 4, m = 2$; $L_1 : 2, 9$; $L_2 : 1, 4$; $L_3 : 5, 6$; $L_4 : 3$.

Example: merge sort

Example

Sort $2, 9, 1, 4, 6, 5, 3$.

Solution

- *Round* 1: $n = 7, m = 3$; $L_i : a_i$ *for* $1 \leq i \leq 7$.
- *Round* 2: $n = 4, m = 2$; $L_1 : 2, 9$; $L_2 : 1, 4$; $L_3 : 5, 6$; $L_4 : 3$.
- *Round* 3: $n = 2, m = 1$; $L_1 : 1, 2, 4, 9$; $L_2 : 3, 5, 6$.

Example: merge sort

Example

Sort $2, 9, 1, 4, 6, 5, 3$.

Solution

- *Round* 1: $n = 7, m = 3$; $L_i : a_i$ *for* $1 \leq i \leq 7$.
- *Round* 2: $n = 4, m = 2$; $L_1 : 2, 9$; $L_2 : 1, 4$; $L_3 : 5, 6$; $L_4 : 3$.
- *Round* 3: $n = 2, m = 1$; $L_1 : 1, 2, 4, 9$; $L_2 : 3, 5, 6$.
- *Round* 3: $n = 1$; $L_1 : 1, 2, 3, 4, 5, 6, 9$.

The complexity

Suppose $n = 2^k$, then there are k rounds. In the i-th round, we have 2^{k+1-i} lists with size 2^{i-1}, and we merge sort them into 2^{k-i} lists with size 2^i. The total number operations in the i-th round is $2^k - 2^{k-i}$. So the total number is

$$\sum_{i=0}^{k-1} (2^k - 2^{k-i}) = (k-1)2^k + 1.$$

The complexity

Suppose $n = 2^k$, then there are k rounds. In the i-th round, we have 2^{k+1-i} lists with size 2^{i-1}, and we merge sort them into 2^{k-i} lists with size 2^i. The total number operations in the i-th round is $2^k - 2^{k-i}$. So the total number is

$$\sum_{i=0}^{k-1} (2^k - 2^{k-i}) = (k-1)2^k + 1.$$

In addition, in each round, there are three more comparisons: $n = 1$? $n = 2m$? $F = 0$? So the complexity function is $(k-1)2^k + 3k + 1 = \mathcal{O}(k2^k) = \mathcal{O}(n \log n)$.

The complexity

Suppose $n = 2^k$, then there are k rounds. In the i-th round, we have 2^{k+1-i} lists with size 2^{i-1}, and we merge sort them into 2^{k-i} lists with size 2^i. The total number operations in the i-th round is $2^k - 2^{k-i}$. So the total number is

$$\sum_{i=0}^{k-1} (2^k - 2^{k-i}) = (k-1)2^k + 1.$$

In addition, in each round, there are three more comparisons: $n = 1$? $n = 2m$? $F = 0$? So the complexity function is $(k-1)2^k + 3k + 1 = \mathcal{O}(k2^k) = \mathcal{O}(n \log n)$.

Remark

$\mathcal{O}(n \log n)$ *is the best complexity for sorting algorithms.*

Homework Assignment #7 - today

Section 8.3 Exercise 9, 10(c), 14, 24.