Introduction to Algorithms

CELEN086

Seminar 5
(w/c 04/11/2024)

Outline

In this seminar, we will study and review on following topics:

- Linear search and Binary search
- Insertion sort
- Time complexity and Big O notation

You will also learn useful Math/CS concepts and vocabularies.

Group activity

- Explain why the linear search has time complexity $O(n)$ and binary search has time complexity $O(\log_2 n)$.

- For insertion sort method, give examples of best/worst case with a list of 5 elements from 1 to 5.

  e.g., [1,3,2,4,5], [2,5,3,4,1], [5,3,2,4,1]…
  There are $5! = 120$ different lists consisting of these 5 elements.

  Which of these 120 different lists requires fewest/most operations using the insertion algorithm in Lecture 5?

Big O notation

Use big O notation to describe time complexities of following functions.

$100n + 30n^3 + 2n^2$	$O(n^3)$	
$25\log n + n$	$O(n)$	$\lg n, \log n, \log_2 n$
$3n^2 \lg n$	$O(\mathrm{n}^2 \lg n)$	have same meanings; they are common notations used in different computer science books.
18000	$O(1)$	
$(n + 20)\log_2 n$	$O(n \lg n)$	

Growth rates of basic functions:

$$Constant < \lg n < n < n \lg n < n^2 < \cdots$$

Insertion sort (best case)

Insertion sort has two components in its operations:

- Making comparisons
- Inserting element into the right place

[5,4,3,2,1]	[ ]
[4,3,2,1]	[5]
[3,2,1]	[4,5]
[2,1]	[3,4,5]
[1]	[2,3,4,5]
[ ]	[1,2,3,4,5]

# of comparisons = 4

# of insertion = 5

In general, if the list has length n

Total # of operations

$$= (n-1) + n$$

$$= 2n - 1 \quad = O(n)$$

Insertion sort (worst case)

[1,2,3,4,5]	[ ]
[2,3,4,5]	[1]
[3,4,5]	[1,2]
[4,5]	[1,2,3]
[5]	[1,2,3,4]
[ ]	[1,2,3,4,5]

In general, if the list has length n

Total # of operations

$$= 1 + 2 + 3 + \cdots + (n-1) + n$$

(comparisons) (insertion)

$$= \frac{1}{2}n^2 + \frac{1}{2}n$$

$$= O(n^2)$$

Practice

- Sort the list [4, 2, 6, 5, 13, 8] using insertion sort.

Left list	Right list

- After sorting the above list, search element 5 in the sorted list using binary search.

Target list	Middle element	Comparison

Note: In exams, you need to show such information as necessary steps in solving the problems.

Practice: Binary search

Algorithm: binSearch(x, sortedList)
Requires: a number x and a sorted list
Returns: True if x is in list; False otherwise

```
let n=length(sortedList)
if n==0 // empty list
    Statement 1
elseif n==1 // single element list
    if Condition 2
        return True
    else
        return False
    endif
else // n>1
    let mid=getNth(n/2+1, sortedList) //get the middle element
    if Condition 3
        return True
    elseif x<mid
        return Statement 4 // cut right half and search on the rest of list
    else
        return Statement 5 // cut left half and search on the rest of list
    endif
endif
```

Sub-algorithms used:

- length(list)
- getNth(n, list)
- cut(list, i, j)

Solution

```
let n=length(sortedList)
if n==0
    return False
elseif n==1
    if  x==value(sortedList)
        return True
    else
        return False
    endif
else
    let mid=getNth(n/2+1, sortedList)
    if  x==mid
        return True
    elseif  x<mid
        return  binSearch(x, cut(sortedList, n/2+1, n) )
    else
        return  binSearch(x, cut(sortedList, 1, n/2+1) )
    endif
endif
```

Algorithm: binSearch(x, sortedList)
Requires: a number x and a sorted list
Returns: True if x is in list; False otherwise

Sub-algorithms used:

- length(list)
- getNth(n, list)
- cut(list, i, j)

Solution

Algorithm: binSearch(x, sortedList)
Requires: a number x and a sorted list
Returns: True if x is in list; False otherwise

```
let n=length(sortedList)
if n==0
   return False
elseif n==1
   if x==value(sortedList)
      return True
   else
      return False
   endif
else
   let mid=getNth(n/2+1, sortedList)
   if x==mid
      return True
   elseif x<mid
      return binSearch(x, cut(sortedList, n/2+1, n) )
   else
      return binSearch(x, cut(sortedList, 1, n/2+1) )
   endif
endif
```

This expression works for both odd and even number n.

Sub-algorithms used:

- length(list)
- getNth(n, list)
- cut(list, i, j)

You should trace this algorithm to see how it works.

Practice :

Read the following algorithms and answer questions given in next slide.

```
g(list)
Requires: A list, list
Returns: ???
return f(list, 0, 0)
```

```
f(list, p, c)
Requires: A list list and two numbers p, c
Returns: ???
if isEmpty(list) then
    return c
else
    if value(list) < p then
        return max(c, f(tail(list), value(list), 1))
    else
        return f(tail(list), value(list), c + 1)
    end if
end if
```

Talking about algorithms in last slide. What would it be useful to mention?

1. This algorithm takes a ……. of ………. as argument and returns …………. .
2. The main algorithm g, calls its …… …,… f.
3. The algorithm f, takes three ………….. , a …… and …………………………..
4. On lines 2, 5, and 7 what is c?
5. On line 4 what is p?
6. 6. Why is the algorithm max called in line 5?

7. What is the base case**?**

8. What does line 4 do?

9. What happens on line 5?

10. What happens on line 7**?**

11. 11. Is f recursive?

Algorithm : Cut(L ,I,j)

Algorithm cut(L, i, j)
Requires: a sorted list L and two integers
Returns : a list between index i and j.

If (i >j) then

Talking about algorithms. What would it be useful to mention?

1. This algorithm takes a ....**list....** of ...**numbers**........ as argument and returns ....**the length of the longest ascending sequence of numbers in the list**..... .

2. The main algorithm g, calls its ...**helper**... ...**function**,... f.

3. The algorithm f, takes three ......**arguments**......... , a ...**list of numbers**... and ....**two numbers**................................

4. On lines 2, 5, and 7 what is c? **c is a counter**

5. On line 4 what is p? **p is the previous value(list); i.e. the number preceding the current one.**

6. Why is the algorithm max called in line 5? **max compares values of c; i.e. it compares the lengths of all the ascending sequences and returns the longest.**

7. What is the base case**?** **When the list is empty c is returned.**

8. What does line 4 do? **Compares a number in the list with the previous number to check whether the sequence is still ascending.**

9. What happens on line 5? **The ascending sequence is broken , so the current value of c is compared with others and the counter c is reset to zero.**

10. What happens on line 7**?** **The sequence is still ascending so, p is changed to the current value for comparison with the next and the counter c is increased by one.**

11. Is f recursive? **Yes – it calls itself in lines 5 and 7.**