FIT1045/FIT1053 Algorithmic Problem Solving – Tutorial 06.

Objectives

After this tutorial, you should be able to:

- Describe computational complexity and Big-O.
- Identify the computational complexity of an algorithm.
- Describe the complexity of different algorithms from this unit.
- Introduce binary search.

Prepared Question

Show the state of the list after each loop when sorting the following lists:

1. $3, 1, 4, 5, 2$ using selection sort.
2. $3, 2, 1, 5, 4$ using insertion sort.

To help you, each of these states should appear exactly once:

- $1, 2, 3, 5, 4$
- $2, 3, 1, 5, 4$
- $1, 2, 3, 4, 5$
- $1, 2, 3, 5, 4$
- $1, 2, 3, 4, 5$
- $1, 2, 3, 5, 4$
- $1, 3, 4, 5, 2$
- $1, 2, 4, 5, 3$

The best case time complexity for insertion sort is O(n), what state do the elements of the list have to be before entering the insertion sort algorithm (i.e. non-decreasing, reversed or random) in order for the algorithm to be O(n).
The best and worst cases for selection sort are both $O(n^2)$ no matter if the list is already sorted, decreasing or randomised before entering the algorithm. Explain why this is the case?

Workbook

§10 Computational Complexity

While your tutors mark the prepared question, attempt the above Workbook sections. Your tutors will go through some of these questions with the class.

Complexity and Big-O

Definition: *The computational (time) complexity of an algorithm is the number of elementary steps $T(n)$ needed for computing its output for an input of a size n.*

Count the number of steps taken by the following programs using the model for counting elementary steps given in lectures. (Start by counting the number of steps taken when $n = 1$ and $n = 2$ by each of these programs.)

1.
```
def program3(n):
    count=0
    i=0
```

```
                 → O(n
    while i<n:
        j=0          → O(n^2
        while j<n:
            count+=1
            j+=1
        i+=1
    return count
```

2. def program4(n):
```
    #get a rough estimate              n      i
    count=0                            5      5
    i=n                                       2
    while i > 0:     → O(logn)
        count+=i
        i//=2
    return count
```

3. def program5(n):
```
    count=100
    i=0
    while i<n:       → O(n)
        j=i
        while j<n:       → O(n^2)
            count*=3
            j+=1
        i+=1
    return count
```

Definition: *A function $t(n)$ is in $O(g(n))$ if there are positive numbers c and n_0 such for all $n \geq n_0$ it holds that $t(n) \leq cg(n)$.*

Discuss this definition. State the complexity of programs 1 to 5 in Big-O notation. For each program, determine c and n_0.

Python Complexity

Many algorithms have a best case and a worst case time complexity. For example, the Python built-in function `any(iterable)`, which returns `True` if any item is true, otherwise it returns `False`, will in the best case only have to check one element. This means that the *best case time complexity* of `any` given in Big-O notation is $O(1)$. How many elements will `any` have to check in the worst case?

For each of the following Python functions, methods, and comparisons, state their best and worst case time complexity in Big-O notation, along with a justification:

In class:

- `x in a_string` when given a character x and a string of length n B: 1 W = n
- `a_string == b_string` when given strings both of length n B = 1 W = n
- `lst[i]` when given a list of length n B: 1 W = 1
- `lst[i:j]` when given a list of length n B: len(lst[i:j]) $\rightarrow O(j-i)$
- `lst.pop()` when given a list of length n $O(1)$
- `lst.append(x)` when given a $\overset{n}{\text{single}}$ item $O(1)$
- `lst.pop(0)` when given a list of length n $\rightarrow O(n)$

add at end

O(n) if lst = n

0; update all

lst.append([1,2,3]) → 3 O(n)

lst += lst 1 [O(1)]

lst = lst + lst 1 O[n]

Practice:

- `all(lst)` when given a list of length n $O(1)$, $O(n)$ ← o False
- `max(lst)` when given a list of length n $O(n)$
- `reversed(lst)` when given a list of length n $O(n)$
- `set(lst)` when given a list of length n
- `lst.remove(x)` when given a list of length n $O(n)$
- `lst.insert(i,x)` when given a list of length n $O(1)$, $O(n)$

end beg

Decrease and Conquer

Discuss the features of a decrease-and-conquer algorithm. What kinds of problems are decrease-and-conquer algorithms used for? What are the limitations of decrease-and-conquer algorithms?
Can you think of some instances of problems in real life where you use a decrease-and-conquer algorithm?
What are some of the problems that computer scientists are interested in for which a decrease-and-conquer algorithm 'works'?

Binary Search

You must guess the number your partner has selected from the numbers 1 to 100. Each time you guess, your partner tells you if you are correct, or if the number you guessed is too large or too small. What is the maximum number of guesses you need to find the number?

Discuss a strategy for solving this problem. Afterwards, implement your strategy in Python.