Data Structures and Algorithms

Prof. Ganesh Ramakrishnan,
Prof. Ajit Diwan,
Prof. D.B. Phatak

Department of Computer Science and Engineering
IIT Bombay

Session: Running Time of Program: Empirical and Analytical Method

Running Time of Programs : Canonical example (searching e in $\underbrace{[\text{"e"}]}_{S}$)

- Consider algorithms for searching for an element e in a sequence S sorted
- Running time is function of ?
  - Input Size
  - Position of e
- Empirically finding running time?

Running Time of Programs

- Consider algorithms for searching for an element e in a sequence S
- Running time is function of ?
  - Input Size
  - Position of e

  } different input parameters
- Empirically finding running time?
  - Run the program (with properly designed running time experiments) multiple times with a large number of different inputs (Sequence S, element e to search for)
  - Time each run and compute average times across runs

Empirical Analysis of Running Time

Need to normalize extraneous factors

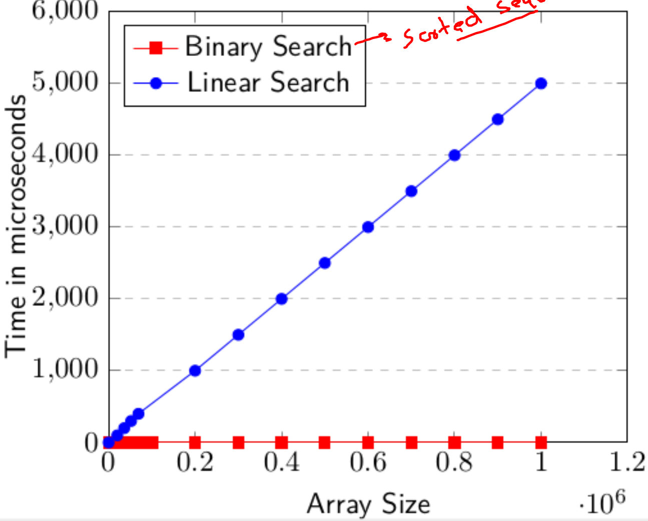
- No background processes
- Comparable data types (structures)
- Single threaded / no wasted steps
- To normalize w.r.t OS/Kernel related background processes
  - Average across multiple runs for each program
  - Use virtual machines with guaranteed resources

int. float

int[ ]

not comparable

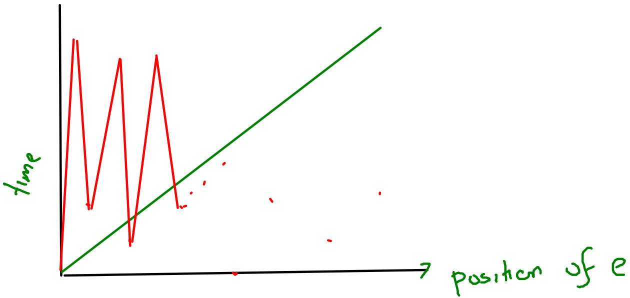
Results of Experiments (Execution time comparison)



→ sorted sequence

Question

How will plot(s) of time vs position of element e being searched for look like (for fixed size of S) ?



Issues with Empirical Analysis

(Post hoc)

- Primary Issue: Reasoning about the outcome should precede the experimentation
- Too time consuming
- Affected by several hidden factors (the hardware, the compiler, etc.)
- Might ignore the essential behaviour of the algorithm while focusing on unnecessary details
- Analysis on paper provides the foundational understanding of a system

Caveat: Such analysis always requires a model of the system under study

Model for Algorithm Analysis

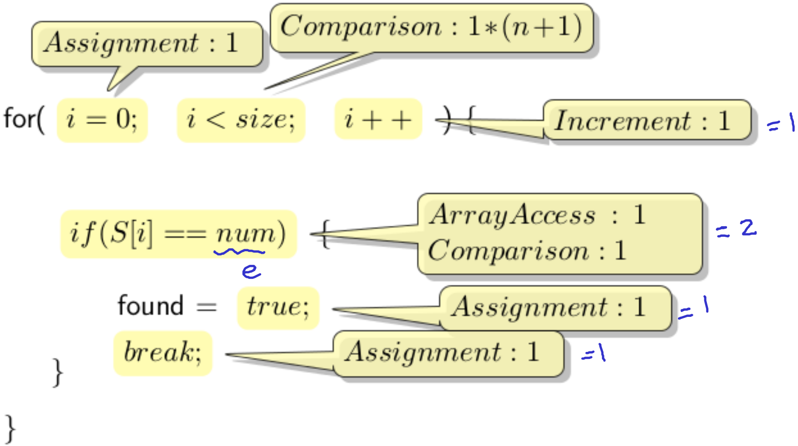
kx → shifting x to left by k positions

$2x$ → shifting x to left

- Sequential instructions and Infinite memory
- All basic instructions take one unit of time - addition, subtraction, multiplication, division, bit operations, comparison, assignment, etc.
- The above assumptions hold equally well for integer and floating point data types
- Ignores disk read times, memory hierarchies, paging, context switching, etc.

Analysis of Linear Search Algorithm A: Successful Search

LinearScan(e, S)



```
for( $i = 0$; $i < size$; $i++$ ){
    if$(S[i] == \underbrace{num}_{e})$ {
        found = $true$;
        $break$;
    }
}
```

$Assignment : 1$

$Comparison : 1*(n+1)$

$Increment : 1$ =1

$ArrayAccess : 1$
$Comparison : 1$ =2

$Assignment : 1$ =1

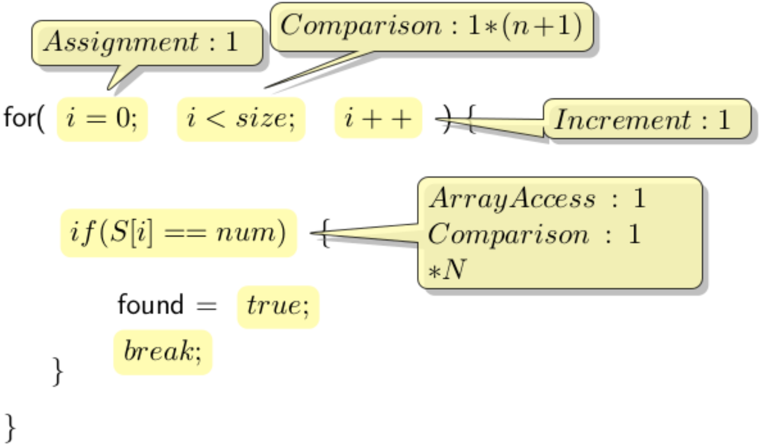
$Assignment : 1$ =1

Search Algorithm A: Time for Successful Search

- Total time $T_s(n)$, when element is at index $n = 0 \ldots N-1$ (successful search)
- $\underline{1} + \underline{2} * (n+1) + \underline{1} + \underline{1} * n + 1 * (n+1)$

  increment
- $T_s(n) = \underline{4n+5}$

Analysis of Search Algorithm A: Unsuccessful Search



```
for( i = 0;   i < size;   i++ ) {

    if(S[i] == num) {

        found = true;
        break;
    }

}
```

$Assignment : 1$

$Comparison : 1*(n+1)$

$Increment : 1$

$ArrayAccess\ : 1$
$Comparison\ : 1$
$*N$

Search Algorithm A: Time for Unsuccessful Search

- Total time for unsuccessful search
- $1 + 2 * N + 1 * N + 1 * (N + 1)$
- $T_u(N) = \underline{4N +} 2$

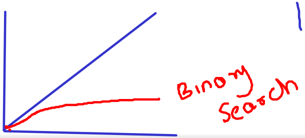
Question

How would you compute the average number of instructions executed by program A?

Question

How would you compute the average number of instructions executed by program A?

Here averaging is across all possible values of the position of ‘$\underbrace{\text{num}}_{(e)}$’ while holding the $\underbrace{\text{length of the list}}_{|s|}$ as a constant



Thank you