Algorithms

Sunday, 22 May 2022 21:48

Simon Hanisch

simonph@outlook.ch

Algorithms

What is an algorithm?

Wikipedia says:

- A **PROCEDURE** (also called a process, method, routine, subroutine, function, or procedure) A finite set of well-defined instructions
- **INPUT** (0 or more) Zero or more quantities that are fed into the algorithm
- **OUTPUT** (1 or more) At least one quantity is produced, having a specified relation to the inputs
- **FINITENESS** The algorithm must always terminate after a finite number of steps
- **ALGORITHM**
  - Step sequence is **DETERMINISTIC**
  - Each step has a well-defined, finite meaning
  - Step 1: Input n
  - Step 2: If $n = 0$ then halt
  - Step 3: If n is divisible by 2 then $n \leftarrow n/2$
  - Step 4: If n is odd then $n \leftarrow 3n + 1$
  - Step 5: Go to step 2
- **EFFECTIVENESS** Operations must be feasible (doable) in principle, by a person using only pencil and paper

Example: Find largest number

- **PROBLEM** Given a list of positive numbers, find the largest one
- **INPUT** List L of positive numbers, $L = [L_0, L_1, L_2, \ldots, L_{n-1}]$
- **OUTPUT** Number m such that $m \geq L_i$ for all i in $0 \ldots n-1$
- **ALGORITHM**
  - Step 1: Input n, L
  - Step 2: $m \leftarrow L_0$
  - Step 3: If $n = 1$ then halt
  - Step 4: $i \leftarrow 1$
  - Step 5: If $L_i > m$ then $m \leftarrow L_i$
  - Step 6: $i \leftarrow i + 1$
  - Step 7: If $i < n$ then go to step 5
  - Step 8: Output m (contains now the largest element from list L)

finite
$n \in \mathbb{N}$
finite order
well-defined
deterministic

$L = [L_0, L_1, L_2, \ldots, L_{n-1}]$
$m \leftarrow L[0]$
for $(i = 1$ to $n-1)$ {
  if $(L[i] > m)$ $m \leftarrow L[i]$
}

Implementation

```
function findMax(int[] list) {
  int max = list[0];
  for (int i = 1; i < list.length; i++) {
    if (list[i] > max) {
      max = list[i];
    }
  }
  return max;
}
```

Complexity

- Time complexity
- Does much training doesn't exist?
- Measure n number of elements
- Measure C_{cmp} number of steps
- Measure C_{mem} number of stores

$n \sim$ size of input

Example analysis

```
function findMax(int[] list) {
  int max = list[0];
  for (int i = 1; i < list.length; i++) {
    if (list[i] > max) {
      max = list[i];
    }
  }
  return max;
}
```


Worst case

depent L (can of any)

Same, but on algorithm?

- **ALGORITHM**
  - Step 1: Input n, L
  - Step 2: If $n = 0$ then halt
  - Step 3: If n is divisible by 2 then $n \leftarrow n/2$
  - Step 4: If n is odd then $n \leftarrow 3n + 1$
  - Step 5: Go to step 2
- Step 1: Input n, L
- Step 2: If $n = 1$ then halt
- Step 3: $i \leftarrow 1$
- Step 4: If $L_i > m$ then $m \leftarrow L_i$
- Step 5: $i \leftarrow i + 1$
- Step 6: If $i < n$ then go to step 4
- Step 7: Output m (contains now the largest element from list L)

Big-O notation

- Mathematical notation
- Asymptotic
  - Big-O
  - Big-Omega
  - Big-Theta
  - Little-o
  - Little-omega
  - Little-theta
- Used in computer science to describe performance or complexity of an algorithm
- Describes worst-case scenario — NP-hard

$2 \cdot 3 = O(n)$

$\log(2^k) = k$

$\log_2 n = k$

$2^k = n$

Big-O visualized

Algorithms

Searching

Check if an element exists in array & return its index

Idea: go through list L and take element x
$L[i] \rightarrow$ compare x & $L[i]$
from start to the finish

$O(n)$

Unsorted List

```
function findMax(int[] list) {
  int max = list[0];
  for (int i = 1; i < list.length; i++) {
    if (list[i] > max) {
      max = list[i];
    }
  }
  return max;
}
```

- Conclusion

Binary Search

Can we do better on sorted list?

$O(n)$

$O(n/2)$

$O(n/4)$

$O(1)$

Can we do better on sorted list?

25 ?

$O(n/2)$

$O(n/4)$

$O(n/8)$

$O(n/16)$

$O(1)$ — not found

Binary search

- Conclusion?

$O(n)$ vs $O(\log_2 n)$

Sorting

- Theory: $O(n \log n)$
- But how?

$n = 2^k$

$n + 1 \quad n = 2^k$

$n + 1 \quad n_{log} = 2^{k-1}$

$n + 1 \quad n_{log} = 2^{k-2}$

$\vdots$

$n + 1 \quad n_{log} = 2^{k-k} = 2^0$

$O(k \text{ steps})$

$O(\log n)$

$\log 2^k = k$

$\Downarrow$

$O(\log n)$

Selection sort

Successively find smallest element?

- Find smallest element
- Move it to beginning
- Repeat on rest array

Start
End

$\frac{End - Start}{2}$

1.5

Floor
ceil
2

Assumpt.

$R(n) = 2 \cdot n - 3 \cdot n$

$n = 2^k$ Reduce Comp.

Selection sort

$k = n - 1 \ldots$

$\frac{n}{2}$, $k \cdot k$

$= \frac{n \cdot (n-1)}{2}$

$= \frac{n^2 - n}{2} < n^2$

Arithm. Series

$\frac{n^2 - n}{2}$

$O(n^2)$ runtime

$O(n)$ - still long

n from algorithm

Merge sort

Successively merge sorted array?

- Merging two already sorted arrays is fast
- Divide and conquer
  - Divide problem (array) into parts
  - Sort parts, recall array (recursively)
  - Merge sorted parts
  - Stop merge the whole array (recursive)

Merge Sort - Divide

Merge Sort - Conquer

$O(n)$

Split $O(n)$

Merge

$L \quad R$

$(L + R) = O(n)$

$\log_2 n$

$\log_2 n$

$O(n \cdot \log n)$

Sort

① Find smallest in array

$2 \mid 8 \mid 10 \mid \ldots$

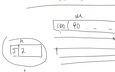
① go through array $O(n)$

② sort + take [0] $O(n \log n)$

Find n smallest in array

$O(1)$

$O(n \cdot m)$



? Data Structure n smallest ?

Heap $\log n$ operation insert a new

$m \cdot \log n$