CS 340 - Analysis of Algorithms Week 2 Lab

Dianna Xu

Example

Employers		Applicants	
A	B	x	y
x	y	B	A
y	x	A	B

- A minimal extreme example where:
  - employers have the exact opposite preferences from the applicants
  - GS matching: [A-x, B-y]
  - What about [A-y, B-x]?
- Employers get the "best possible" matching
- Evaluated *collectively*

2

Best Possible

- a is a *valid match* for e if there is some stable matching that includes the pair (e, a)
- let $best(e)$ return the highest ranked valid match of e
- A matching S^* is "best possible" when
  - $S^* = (e, best(e)), \forall e \in E$

3

Data Structures

- State your data structure choices
- Reason how specific data structure construction/update/access times affect your time analysis

	Unsorted array	Sorted array	Unsorted list	Sorted list	Balanced BST
search	$O(n)$	$O(logn)$	$O(n)$	$O(n)$	$O(logn)$
insert	$O(1)^*$	$O(n)$	$O(1)$	$O(n)$	$O(logn)$
remove	$O(1)^*$	$O(n)$	$O(1)$	$O(1)$	$O(logn)$
min/max	$O(n)$	$O(1)$	$O(n)$	$O(1)$	$O(logn)$

4

Understanding Efficiency

- Goal: Algorithms should be fast and not use too much space
- We'll concentrate mostly on time analysis
- What does this mean quantitatively?
  - not dependent on platforms, problem instances, input sizes

Asymptotic Notation

- Provides a way to simplify analysis
- Allows us to ignore less important elements
  - constant factors
- Focus on the largest growth of n
- Focus on the dominant term
- We measure time complexity with the input size, i.e. n of the $O(n)$
  - n is just a variable, it can be a function of any complexity

6

How do these functions grow?

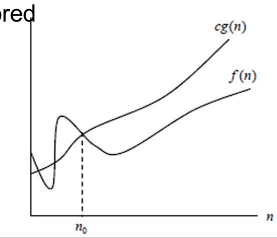
- $f_1(x) = 43x^2 \log^4 x + 12x^3 logx + 52xlogx$
- $f_2(x) = 15x^2 + 7xlog^3 x$
- $f_3(x) = 3x + 4\log_5 x + 91x^2$
- $f_4(x) = 13 \cdot 3^{2x+9} + 4x^9$
- $f_5(x) = \sum_{x=0}^{\infty} \frac{1}{2^x}$

7

Big O

$\exists\, n_0 \geq 0, c > 0$, if $f(n) \leq c \cdot g(n) \forall n \geq n_0$, then $f(n) = O(g(n))$

- Constant factors are ignored
- Upper bound
- $\lim_{n\to\infty} \frac{f(n)}{g(n)} \leq c$



Gale-Shapely is Linear!

- Input
  - A set of employers: E
  - A set of applicants: A
  - $|E| = |A| = n$
  - $2n$ preference lists, each of size n
- Size of input: $2n^2$
- Let $N = 2n^2$
- while loop runs n^2 iterations
- $n^2 = O(2n^2) = O(N)$

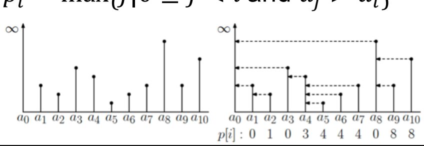
Read and Review

- Chapter 2 – review heaps
- Chapter 3
  - BFS
  - DFS
  - Properties
  - Implementations
- Review binary search trees
  - AVL or Red/Black
  - search
  - insertion
  - deletion
  - traversal

10

Previous Greater Element

- A list of numeric values $\langle a_1, a_2, \ldots, a_n \rangle$
- For each a_i, find the index of the rightmost element of the sequence $\langle a_1, a_2, \ldots, a_{i-1} \rangle$ whose value is strictly greater than a_i, or 0
- or $a_0 = \infty$
- $p_i = \max\{j | 0 \leq j < i \text{ and } a_j > a_i\}$



11

Worst-case Analysis

- Dominated by the time spent in the inner loop
- $T(n) = \sum_{i=1}^{n} \sum_{j=0}^{i-1} 1$
- $= 1 + 2 + \cdots (n-2) + (n-1)$
- $= \sum_{i=1}^{n-1} i$
- $= \frac{(n-1)n}{2}$
- $\in \Theta(n^2)$

12

Improvement?

13

Naïve Solution

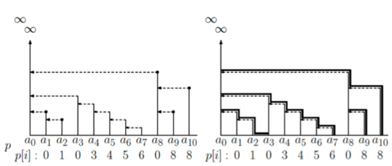
```
//input:  an  array  of  numeric  values  a[1..n]
//returns: an array p[1..n] where p[i] stores the
//index of the previous larger element of a[i]

PL(a) {
  for (i = 1 to n) {
    j = i-1
    while (j>0 and a[j] <= a[i])
      j--
    p[i] = j
  }
  return p
}
```

14

Improvement

- When computing p_i, we already know the values of p_1 to p_{i-1}
- Use p values to leapfrog the inner loop



15

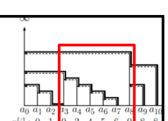
Improved Solution

```
//input:  an  array  of  numeric  values  a[1..n]
//returns: an array p[1..n] where p[i] stores the
//index of the previous larger element of a[i]

PL(a, n) {
  for (i = 1 to n) {
    j = i-1
    while (j>0 and a[j] <= a[i])
      j = p[j]
    p[i] = j
  }
  return p
}
```

16

Is it really better?



- $a_j, \dots, a_{i-1}$ in decreasing order, a_i is larger than all: $(a_3, \dots, a_8)$ in the example
  - processing a_i will force $O(i - j - 1)$ steps
  - can this happen all the time?
    - $a_{i+1} \geq a_i$
    - $a_{i+1} < a_i$
  - once p_i is set, we never visit $a_j, \dots, a_{i-1}$ again – at most n
  - or `while` is not executed at all – at most n
- $O(n)$

17