Lec_35.md

Lecture 35

Dec. 08/2020

Announcements

Final Exam

- Summative Assessment
- 35% of ECE244 Mark
- December 14, 2020
  - Starting at 9:30am ET
  - 2.5 hour duration
  - Synchronous Exam
    - Everyone takes exam at same time, regardless of time zone
- Open textbook, open notes
- No calculator, No IDE's
- No more than 50% MC type questions

Optional Lab

- 12 students completed and competed in the lab

Teaching Evaluations

Complexity Analysis

We can measure the complexity of an algorithm in time versus the input size of the problem

- Group the complexity of algorithm's into **complexity classes**

Complexity Classes

Complexity Classes:

1. `O(1)`

- This is rare
- Same runtime regardless of how large the size of the input is

2. `O(log_n)`

- Great algorithms

3. `O(n)`

- Linear algorithms

4. `O(nlogn)`

- Somewhere between linear and n-squared

5. `O(n^2)`

- Polynomial growth

6. `O(n^3)`
7. `O(2^n)`
8. `O(n!)`

- The dreaded "factorial time"

9. `O(n^n)`

Recursive Functions

What would the **time complexity** be for a recursive function?

For example,

```
int recursive(int n){
  if(n<1) return 1;
  else return (recursive(n/2));
}
```

Notes:

1. `if(n<1) return 1;`

- This is a step!

2. T(n) = **recurrence equation**

- C, if n<1
- C+T(n/2), if n>1

T(n)
T(n)=C+T(n/2)
T(n)=C+T(n/4)
T(n)=C+C+T(n/8)
T(n)=C+C+C+T(n/16)
T(n)=C+C+C+...+T(n/n)

Notice that this is proportional to log_2(n)

- Require *n* divisions to reach T(n/n)

So `int recursive()` has **time complexity** `O(logn)`

```
int recursive(int n){
  if(n<=1) return 1;
  else return (recursive(n-1)+recursive(n-1));
}
```

Notes:

1. `if(n<=1) return 1;`

- This is a step

2. T(n) = **recurrence equation**

- C, if n<=1
- C+2T(n-1), if n>1

T(n)	
T(n)=C+2T(n-1)	T(n)=C+2T(n-1)
T(n)=C+2(C+2T(n-2))	T(n)=3C+4T(n-2)
T(n)=C+2(C+2(C+2T(n-3)))	T(n)=7C+8T(n-3)
T(n)=...	T(n)=15C+16T(n-4)
T(n)=xC+yT(n-(n-1))	T(n)=(2^(n-1)-1)C+2^(n-1)T(1)

Notice that in `T(n)=(2^(n-1)-1)C+2^(n-1)T(1)` , `T(1)=C`

- So the time complexity of the algorithm is `T(n)` is proportional to `2^(n-1)C`
  - So the order of this algorithm is `O(2^n)`

```
int recursive(int n){
  if(n<=1) return 1;
  else return (recursive(n/2)+recursive(n/2));
}
```

Notes:

1. T(n) =

- C, if n<=1
- C+2T(n/2), if n>1

T(n)
T(n)=C+2T(n/2)
T(n)=3C+4T(n/4)
T(n)=7C+8T(n/8)
T(n)=15C+16T(n/16)
T(n)=(n-1)C+nT(n/n)

Quicksort

```
int QuickSort(int left,int right,int* a){
  int p = SelectAndShuffle(left, right, a);
  if(p>left) QuickSort(left,p-1,a);
  if(p<right)QuickSort(p+1,right,a);
}
```

Notes:

1. `SelectAndShuffle`

- **time complexity**: `O(n)`

2. `QuickSort(left,p-1),a);` and `QuickSort(p+1,right,a)`

- T(n) = n + T(p) + T(n-p)
  - The **time complexity** of this *depends* on input data
    - Location of pivot with respect to data in `a`
    - Split into best, worst, and average case