# Priya Loran - CSCI 104 HW1

## Problem 1: Runtime Analysis

Submit .pdf (scanned)
hw1_answers.pdf
Function of n, input

### a

* Operations take constant time (c)
* while loops have # of iterations
* Focus on dominating terms

```
void f1(int n)
{
    int i=2;
    while(i < n){
        /* do something that takes O(1) time */
        i = i*i;
    }
}
```

c
c
c

Doing this — iterations → Requires calculation

Runtime: $c\left[\underbrace{2<n,\ 2^2<n,\ 4^2<n,\ 16^2<n\ldots}\right]$

Go until $(2^x)^2 \geq n$ where $x = 0,1,2\ldots$

Solve $(2^x)^2 \geq n$ for $x$ we get $(2^x)^2 \geq n = 2^{2^k} \geq n$

$\log 2^{2^k} \geq \log n \rightarrow 2^k \log 2 \geq \log n \rightarrow 2^k > \log n \rightarrow \log 2^k \geq \log n$

$k \log 2 \geq \log n \rightarrow k \geq \log(\log n)$

$\boxed{\therefore \text{ the runtime is } O(\log(\log n))}$

### b

* If loop conditions depend on others → Summation
* "If-branches" assume worst case and must use algebra to predict # of times it runs

```
void f2(int n)
{
    for(int i=1; i <= n; i++){
        if( (i % (int)sqrt(n)) == 0){
            for(int k=0; k < pow(i,3); k++) {
              /* do something that takes O(1) time */
            }
        }
    }
}
```

C — Do this n iterations — Do this when i is a multiple of $\sqrt{n}$ — Requires Summation Calculation

How many times does "#" run? :

Say $n=25 \to \sqrt{n}=5 \to 5$ multiples $(5,10,15,20,25 = i)$

Say $n=9 \to \sqrt{9}=3 \to 3$ multiples $(3,6,9 = i)$

Say $n=30 \to \sqrt{30}=5.48 \to \text{int}=5 \to 6$ multiples

$\boxed{\therefore \text{ can equal } n \ (\leq)}$

So the branch seems to run $\frac{n}{\text{int}(\sqrt{n})}$ iterations $= \sqrt{n}$ iterations

How many times does inner loop run?

- From $k=0$ to $i^3$ where $i$ = multiple of $\sqrt{n}$

Let a multiple of $\sqrt{n}$, $i$, be written as $x \cdot \sqrt{n}$ where $x = 1, 2, 3 \ldots \sqrt{n}$ $\therefore i = x \cdot \sqrt{n}$

upper bound =

$\sqrt{n}\, x \geq n$

$x \geq \frac{n}{\sqrt{n}}$

$x \geq \sqrt{n}$

- The # of iterations "if" is true is $\sqrt{n}$
- The inner loop runs $i^3$ times $= (x \cdot \sqrt{n})^3$ times for $\sqrt{n}$ times

Total Summation =

$$\sum_{x=1}^{\sqrt{n}} (i)^3 = \sum_{x=1}^{\sqrt{n}} (x \cdot \sqrt{n})^3 = \sqrt[3]{n^2} \sum_{x=1}^{\sqrt{n}} x^3 = \sqrt[3]{n^2} \left[\frac{\sqrt{n}(\sqrt{n}+1)}{2}\right]^2 =$$

$$n^{3/2}\left[\frac{(n+\sqrt{n})(n+\sqrt{n})}{4}\right] = n^{3/2}\left[\frac{n^2+n\sqrt{n}+n\sqrt{n}+n}{4}\right] \to n^{3/2} \cdot n^2$$

Note:

$$\sum_{a=1}^{t} x^3 = \frac{t^4}{4}$$

*common summation

*only care about asymptotic behavior so use $n^2$, drop $\frac{1}{4}$

$$= n^{3/2} \cdot n^{4/2} = n^{7/2}$$

$\boxed{\therefore \text{ the runtime is } O(n^{7/2})}$

## c

* If loop conditions are independent → Multiplication
* In worst case, assume all array elements = i

```cpp
for(int i=1; i <= n; i++){
  for(int k=1; k <= n; k++){
    if( A[k] == i){
      for(int m=1; m <= n; m=m+m){
        // do something that takes O(1) time
        // Assume the contents of the A[] array are not changed
      }
    }
  }
}
```

c · Do this n iterations · Do this n iterations · Do this when array element = i · Do this $2^m = n$ → logn iterations

How many times does this run? : Loop runs while $m \leq n$ → $1, 2, 4, 8 \ldots 2^m$ where $2^m \leq n$ in worst case $2^m = n$

$2^m = n \rightarrow \log_2 n = m$

$c[n \cdot n \cdot \log n] = [n^2 \log n] c$ → Exponential · Logarithmic · Constant

* Exponential grows fastest, then logarithmic, but constant has limited effect

$cn^2 \log n$ is $O(n^2 \log n)$ ∴ the runtime is $O(n^2 \log n)$ in worst case

* In best case, assume NO elements = i

- "if" branch never runs; only the first two for loops

$n \cdot n = n^2$ ∴ the runtime is $O(n^2)$ in best case

## d

```cpp
int f (int n)
{
  int *a = new int [10];
  int size = 10;
  for (int i = 0; i < n; i ++)
     {
        if (i == size)
          {
             int newsize = 3*size/2;
             int *b = new int [newsize];
             for (int j = 0; j < size; j ++) b[j] = a[j];
             delete [] a;
             a = b;
             size = newsize;
          }
        a[i] = i*i;
     }
}
```

O(newsize) · O(size) · c · c · c · c · c · c · c · O(1)·n

Do this n iterations · To determine # of times "if" runs, do calculations · Requires calculations

- when $i = size \rightarrow$ run
- $i$ increases by 1 $\rightarrow$ Go until $i = n$
- $size = newsize$, $size = \frac{3size}{2}$

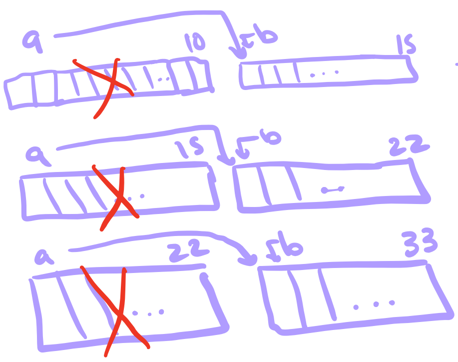
$$= \underbrace{n + 10 + \left(\tfrac{3}{2}\right)10 + \left(\tfrac{3}{2}\right)^2 10}\ + \dots \left(\tfrac{3}{2}\right)^{\log n} 10$$

So "if" runs approx. $\log n$ iterations

$$\left(\tfrac{3}{2}\right)^x 10 \geq n = \left(\tfrac{3}{2}\right)^x \geq \frac{n}{10} = x\log\left(\tfrac{3}{2}\right) \geq \log\left(\tfrac{n}{10}\right)$$

*Take out constants $\left(\frac{1}{\log 3/2}\right)$

$$x = \frac{\log \frac{n}{10}}{\log \frac{3}{2}} \rightarrow x \text{ is } O(\log n)$$




$\rightarrow$ if $i = 10$

$\rightarrow$ if $i = 15$

$\rightarrow$ if $i = 22$

= newsize (Dynamic Array allocation) + size (copy array)

= $O(newsize + size)$ but it goes until $size == i == n$ so

*Geometric Summation

$$\text{Runtime} = n + 10\sum_{k=0}^{\log n}\left(\tfrac{3}{2}\right)^k = n + 10\left[\frac{1-\left(\frac{3}{2}\right)^{\log n}}{1-\frac{3}{2}}\right]$$

*Take out constants

$$\text{Runtime} = n + \left(\tfrac{3}{2}\right)^{\log n} = n + e^{\log n \log \frac{3}{2}} = n + n^{\log \frac{3}{2}}$$

$n^{\log \frac{3}{2}}$ grows slower than $n$

$$\boxed{\therefore \text{ the runtime is } O(n)}$$

# Problem 2 - LL Recursion Tracing

## a

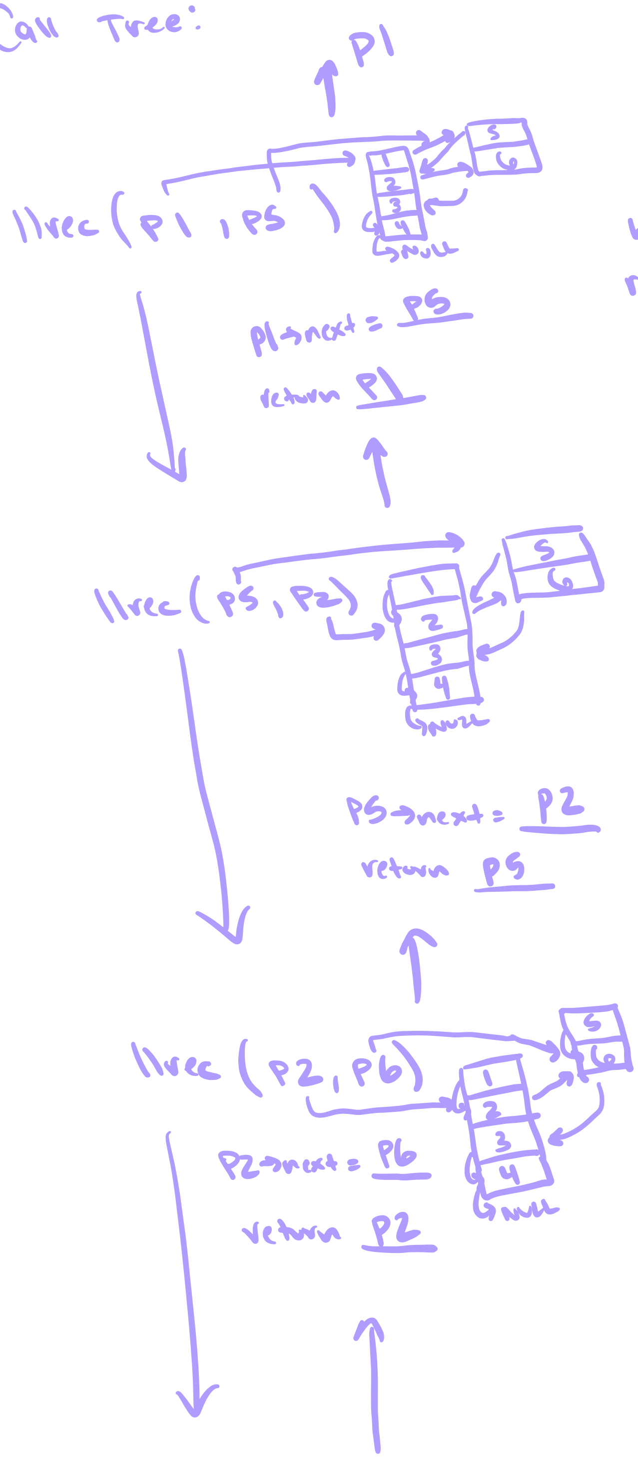
```
struct Node {
    int val;
    Node*  next;
};

Node* llrec(Node* in1, Node* in2)
{
    if(in1 == nullptr) {
        return in2;
    }
    else if(in2 == nullptr) {
        return in1;
    }
    else {
        in1->next = llrec(in2, in1->next);
        return in1;
    }
}
```

in1 = 1,2,3,4

in2 = 5,6

Call Tree:

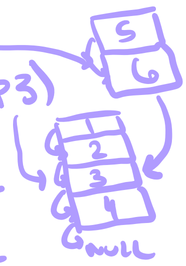



With this input, the program returns a pointer to the first address of a linked list with value of 1. The LL goes

1→5→2→6→3→4→NULL

- After changing pointers, I also changed all same pointers above that function
- Arrows are where node→next pointer points to
- Px is a pointer to a linked list with a value (int) of x

llrec(P6, P3)




P6→next = $\underline{P3}$

return $\underline{P6}$

llrec(P3, NULL)

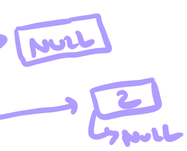
return $\underline{P3}$

**b**

Can Tree:

↑ P2

llrec(NULL, P2)




return $\underline{P2}$

```
struct Node {
    int val;
    Node*  next;
};

Node* llrec(Node* in1, Node* in2)
{
    if(in1 == nullptr) {
        return in2;
    }
    else if(in2 == nullptr) {
        return in1;
    }
    else {
        in1->next = llrec(in2, in1->next);
        return in1;
    }
}
```

in1 = NULL

in2 = 2

With this input, the program returns a pointer to the first address of a linked list with value of 2. The LL goes

2 → NULL

- Px is a pointer to a linked list with a value (int) of x
- Only one call to llrec(Node*, Node*)