getting a runtime function from an algorithm or program.

Basic idea: Count operations:

\[ + \times / = 2 \] = Cost of 1

\[ x[4] = 7 \times y[5] + 4; \]

Cost = 5

Rule 1)

\[ S1; \]

\[ S2; \]

\[ \text{Cost} = \text{cost}(S1) + \text{cost}(S2) \]

Rule 2) if:

\[ \text{if (test)} \]

\[ S1; \]

\[ \text{else} \]

\[ S2; \]

\[ \text{Cost} = \text{cost(test)} + \max(\text{cost}(S1), \text{cost}(S2)) \]

Rule 3) for-loops

\[ \text{for (init, test, update)} \]

\[ S1; \]

\[ \text{Cost} = \text{cost(init)} + \sum_{\text{index} = \text{start}}^{\text{end}} (\text{cost}(S1) + \text{cost(test)} + \text{cost(update)}) \]

Rule 4) while-loops

\[ \text{while (test)} \]

\[ S1; \]

Assume it terminates

Consider worst case number of repetitions, K

\[ \text{Cost} = K \times (\text{cost}(S1) + \text{cost(test)}) \]

\[ 3n^2 + 8 \geq 7n^2 + 8n + 2 \geq 9n^2 + 128 \Rightarrow O(n^2) \]
**Applying to Code  EX #1**

```c
void Arraylist::swap(int i, int j)
T tmp;

tmp = data[i];
data[i] = data[j];
data[j] = tmp;
return;
```

---

**EX #2**

```c
foo(int n, int k)
int x;

if (n == 0)
x = 0;
else {
x = k * k;
x = x / n;
}
return x;
```

---

**EX #3**

```c
sum(int a[], int n)
int s = 0;

for (int k=0; k<n; k++){
s = s + a[k];
}
return s;
```

---

**EX #4**

```c
LL* LinkedList::find( T x )
LL* p = this;

while (p->next != NULL){
    if (p->data == x){
        // code...
    }
    p = p->next;
}
```
```c
while (p->next != NULL){
    if (p->data == x){
        return p;
    }
    p = p->m_next;
}
return NULL;

EX #5

sum_sqr(int a**, int n)
int s = 0;
for (int k=0; k<n; k++)
{
    for(int j=0; j<n; j++)
    {
        s = s + a[k][j];
    }
}
return s;

EX #6

int x = 1;
for (int i=0; i<n; i++)
{
    for(int j=0; j<n*n; j++)
    {
        x = x + k * i;
    }
}

EX #7

log n ?
```
log \_2 n \ ?

- Given a sorted array find(x) in the array?
- How many times can you split an array in half?
- To which power do you raise 2 to match the size of an array?

\[ \log_2 n \quad n: \text{size of array} \]

Apply to List Implementation (Data Structure)

<table>
<thead>
<tr>
<th></th>
<th>ArrayList</th>
<th>LinkedList</th>
<th>Enc. LinkedList</th>
</tr>
</thead>
<tbody>
<tr>
<td>front</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>back</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>at</td>
<td>1</td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td>insert_back</td>
<td>n</td>
<td>n</td>
<td>1</td>
</tr>
<tr>
<td>insert_front</td>
<td>n</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>insert</td>
<td>n</td>
<td>0</td>
<td>( \text{Variable} )</td>
</tr>
<tr>
<td>remove</td>
<td>n</td>
<td>1</td>
<td>( \text{Variable} )</td>
</tr>
<tr>
<td>find</td>
<td>n</td>
<td>n</td>
<td>n</td>
</tr>
</tbody>
</table>

**Importance**

Moore's Law
Wirth's Law: Software becomes slower faster than hardware becomes faster...

Data Structures Page 5
"The best program known to solve such problem has complexity $n^2$.

Sorting: $n^2$, C.A.R. Hoare $n \log_2 n$ "Quicksort"

Matrix multiplication: $n^3$, $n^{2.56}$

SAT $a \lor b \lor c \land (c \lor b) \land (\neg a)$ $2^n$

Cryptography $\text{msg} \Rightarrow \text{cyrkr}$ prime factors of $q$

Prime factorisation $2^n$