Announcements

- Lecture tomorrow
- HW due Monday
- Next program will be up Saturday
Functions

public:

- SLinked List() const;
- ~SLinked List() const;
- bool empty() const;
- const Object& front() const;
- void addFront(const Object& e);
- void removeFront();
remove Front

template <typename Object>
void LinkedList<Object>::removeFront()
{
    SNode <Object> * temp;
    temp = _head;
    _head = _head -> _next;
    delete temp;
}
Destructor

```cpp
template <typename Object>
SLinkedList <Object> :: ~SLinkedList () {
    while (!empty ()) {
        removeFront ();
    }
}
```
template<typename Object>
void SLinkedList<Object>::addFront(const Object &data)
{
    SNode<Object> *newGuy = new SNode<Object>;
    newGuy->elem = data;
    newGuy->next = _head;
    _head = newGuy;
}
Algorithm Analysis (Ch. 4)

How do we compare two programs?

- memory usage
- speed
- interface
- features
- benchmarks
- cost
Speed

How fast an algorithm runs can be very dependent on variables in the system.

Examples:
- hardware
- other software on system
- OS
- language
- input
Primitive Operations

As a way to compare algorithms in a generic way, we instead count primitive operations. Of these:

- addition, subtraction, memory access, return, mult & div

In addition, we (generally) only analyze the worst possible running time.

Why? Guaranteeing a minimum performance.
Comparing

OK, so we have the worst case # of operations - usually a function of \( n \).

How to compare?

- Binary search versus
- Linear search
  - checks every element

\[ 4n \log_2 n \text{ operations} \]
\[ 2n \log_2 n + 1.26Sn \]
Big-O

We say $f(n)$ is $O(g(n))$ if $\forall n > n_0$, there exists $c > 0$ such that $f(n) \leq c \cdot g(n)$.

$f(n) \leq g(n)$

for all
Ex: $5n$ is $O(n^2)$

\[ \forall n \geq 1, \quad 5n \leq 5n^2 \]

Ex: $5n$ is $O(n)$

\[ 5n \leq 5n \]

Ex: $16n^2 + 52$ is $O(n^2)$

\[ 16n^2 + 52 \leq 16n^2 + 52n^2 = 68n^2 \]

\[ \Rightarrow a_nx^n + a_{n-1}x^{n-1} + \ldots + a_0 \text{ is } O(x^n) \]
Functions we will use

1. $O(1)$ - constant time
2. $O(\log n)$ - logarithmic time (by Binary Search)
3. $O(n)$ - linear time
4. $O(n \log n)$
5. $O(n^2)$ - quadratic time
6. $O(n^3)$ - cubic time
7. $O(2^n)$ - exponential time

$log_2 a + log_2 b = log_2(ab)$
$log_2 x^c = c log_2 x$