Homework 3 Answered

All runtime tables, runtime functions and Big–O notations should be in one file submitted to BrightSpace. If done on paper, upload a photo or scan to BrightSpace.

In your GitHub class repository, under HW/Homework03, analyze the following functions from ArrayList.h and LinkedList.h:

ArrayList	LinkedList
resize()	**insert()**
insert()	**insertFront()**
insertFront()	**pop()**
pop()	**popFront()**
popFront()	**remove()**
remove()	**contains()**
contains()	**getSize()**
getSize()	**toString()**
toString()	**reverse()**
reverse()	
find()	

Please provide runtime tables, runtime functions, and the Big–O notation for each of the methods above.

For each please indicate the name of the class and the method. (e.g. ArrayList::resize()).

ArrayList Methods

All costs can be considered 1 and n is the number of elements in the ArrayList object.

```
void ArrayList::resize()
{
     capacity *= 2;
     T* newArr = new T[capacity];
     for (int i = 0; i < size; ++i)
     {
          newArr[i] = arr[i];
     }
     delete[] arr;
     arr = newArr;
}
```

Step	Statement	Cost	Time
1	capacity *= 2;	c_1	1
2	T* newArr = new T[capacity];	c_2	1
3	int i = 0;	c_3	1
4	i < size;	c_4	n + 1
5	newArr[i] = arr[i];	c_5	n
6	++i	c_6	n
7	delete[] arr;	c_7	n + 1
8	arr = newArr;	c_8	1

$T(n) = c_1 + c_2 + c_3 + c_4(n + 1) + c_5(n) + c_6(n) + c_7(n + 1) + c_8$

$= (c4 + c5 + c6 + c7)n + (c_1 + c_2 + c_3 + c_4 + c_7 + c_8)$

$= 4n + 6$

Big-O = O(n)

```
void ArrayList::insert(T value)
{
	if (size == capacity)
	{
		resize();
	}
	arr[size] = value;
	size++;
}
```

Step	Statement	Cost	Time
1	size == capacity	c_1	1
2	resize();	c_2	0
3	arr[size] = value;	c_3	1
4	size++;	c_4	1

$T(n) = c_1 + c_3 + c_4$

$= 3$

Big-O = $O(1)$

Did not count resize() since the function only gets called when the array fills up. Which is not often.

```
void ArrayList::insertFront(T value)
{
    if (size == capacity)
    {
        resize();
    }
    for (int i = size; i > 0; --i)
    {
        arr[i] = arr[i - 1];
    }
    arr[0] = value;
    size++;
}
```

Step	Statement	Cost	Time
1	`size == capacity`	c_1	1
2	`resize();`	c_2	0
3	`int i = size;`	c_3	1
4	`i > 0;`	c_4	n + 1
5	`arr[i] = arr[i - 1];`	c_5	n
6	`--i`	c_6	n
7	`arr[0] = value;`	c_7	1
8	`size++;`	c_8	1

$T(n) = c_1 + c_3 + c_4(n + 1) + c_5(n) + c_6(n) + c_7 + c_8$

$= (c_4 + c_5 + c_6)n + (c_1 + c_3 + c_4 + c_7 + c_8)$

$= 3n + 5$

Big-O = O(n)

Did not count resize() since the function only gets called when the array fills up. Which is not often.

```
T ArrayList::pop()
{
     if (size == 0)
     {
          throw runtime_error("List is empty");
     }
     size--;
     return arr[size];
}
```

Step	Statement	Cost	Time
1	`size == 0`	c_1	1
2	`throw runtime_error("List is empty");`	c_2	0
3	`size--;`	c_3	1
4	`return arr[size];`	c_4	1

$T(n) = c_1 + c_3 + c_4$

$= 3$

Big-O = O(1)

```
T ArrayList::popFront()
{
	if (size == 0)
	{
		throw runtime_error("List is empty");
	}
	T value = arr[0];
	for (int i = 1; i < size; ++i)
	{
		arr[i - 1] = arr[i];
	}
	size--;
	return value;
}
```

Step	Statement	Cost	Time
1	size == 0	c1	1
2	throw runtime_error("List is empty");	c2	0
3	T value = arr[0];	c3	1
4	int i = 1;	c4	1
5	i < size;	c5	n
6	arr[i - 1] = arr[i];	c6	n - 1
7	++i	c7	n - 1
8	size--;	c8	1
9	return value;	c9	1

$$T(n) = c1 + c3 + c4 + c5(n) + c6(n - 1) + c7(n - 1) + c8 + c9$$

$$= (c5 + c6 + c7)n + (c1 + c3 + c4 + c6 + c7 + c8 + c9)$$

$$= 3n + 7$$

Big-O = $O(n)$

```
bool ArrayList::remove(T value)
{
    for (int i = 0; i < size; ++i)
    {
        if (arr[i] == value)
        {
            for (int j = i; j < size - 1; ++j)
            {
                arr[j] = arr[j + 1];
            }
            size--;
            return true;
        }
    }
    return false;
}
```

Step	Statement	Cost	Time
1	int i = 0;	c_1	1
2	i < size;	c_2	0
3	arr[i] == value	c_3	1
4	int j = i;	c_4	1
5	j < size - 1;	c_5	n
6	arr[j] = arr[j + 1];	c_6	n - 1
7	++j	c_7	n - 1
8	size--;	c_8	1
9	return true;	c_9	1
10	++i	c_{10}	0
11	return false;	c_{11}	0

$$T(n) = c_1 + c_3 + c_4 + c_5(n) + c_6(n - 1) + c_7(n - 1) + c_8 + c_9$$

$$= (c_5 + c_6 + c_7)n + (c_1 + c_3 + c_4 + c_6 + c_7 + c_8 + c_9)$$

$$= 3n + 7$$

Big-O = O(n)

```
bool ArrayList::contains(T value) const
{
     for (int i = 0; i < size; ++i)
     {
          if (arr[i] == value)
          {
               return true;
          }
     }
     return false;
}
```

Step	Statement	Cost	Time
1	`int i = 0;`	c_1	1
2	`i < size`	c_2	n + 1
3	`arr[i] == value`	c_3	n
4	`return true;`	c_4	0
5	`++i`	c_5	n
6	`return false;`	c_6	1

$$T(n) = c_1 + c_2(n + 1) + c_3(n) + c_5(n) + c_6$$

$$= (c_2 + c_3 + c_5)n + (c_1 + c_2 + c_6)$$

$$= 3n + 3$$

Big-O = O(n)

```cpp
int ArrayList::getSize() const
{
	return size;
}
```

Step	Statement	Cost	Time
1	`return size;`	c_1	1

$T(n) = c_1$

$\quad\;\; = 1$

Big-O = O(1)

```
string ArrayList::toString() const override
{
     stringstream out;
     out << "[";
     for (int i = 0; i < size; ++i)
     {
          out << arr[i];
          if (i < size - 1)
          {
               out << ",";
          }
     }
     out << "]";
     return out.str();
}
```

Step	Statement	Cost	Time
1	stringstream out	c_1	1
2	out << "[";	c_2	1
3	int i = 0;	c_3	1
4	i < size;	c_4	n + 1
5	out << arr[i];	c_5	n
6	i < size - 1	c_6	n
7	out << ",";	c_7	n - 1
8	++i	c_8	n
9	out << "]";	c_9	1
10	return out.str();	c_{10}	1

$$T(n) = c1 + c2 + c3 + c4(n + 1) + c5(n) + c6(n) + c7(n - 1)$$

$$+ c8(n) + c9 + c10$$

$$= (c4 + c5 + c6 + c7 + c8)n + (c1 + c2 + c3 + c4 - c7 + c9 + c10)$$

$$= 5n + 5$$

Big-O = $O(n)$

```
void ArrayList::reverse()
{
     int left = 0;
     int right = size - 1;
     while (left < right)
     {
          T temp = arr[left];
          arr[left] = arr[right];
          arr[right] = temp;
          left++;
          right--;
     }
}
```

Step	Statement	Cost	Time
1	`int left = 0;`	C1	1
2	`int right = size - 1;`	C2	1
3	`left < right`	C3	$\left\lfloor \frac{n}{2} \right\rfloor + 1$
4	`T temp = arr[left];`	C4	$\left\lfloor \frac{n}{2} \right\rfloor$
5	`arr[left] = arr[right];`	C5	$\left\lfloor \frac{n}{2} \right\rfloor$
6	`arr[right] = temp;`	C6	$\left\lfloor \frac{n}{2} \right\rfloor$
7	`left++;`	C7	$\left\lfloor \frac{n}{2} \right\rfloor$
8	`right--;`	C8	$\left\lfloor \frac{n}{2} \right\rfloor$

$$\begin{aligned} T(n) &= C1 + C2 + C3\left(\left\lfloor \frac{n}{2} \right\rfloor + 1\right) + C4\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + C5\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + C6\left(\left\lfloor \frac{n}{2} \right\rfloor\right) \\ &\quad + C7\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + C8\left(\left\lfloor \frac{n}{2} \right\rfloor\right) \\ &= (C3 + C4 + C5 + C6 + C7 + c8)\left\lfloor \frac{n}{2} \right\rfloor + (C1 + C2 + C3) \\ &= 6\left\lfloor \frac{n}{2} \right\rfloor + 3 \end{aligned}$$

Big-O = O(n)

```
int find(T value) const
{
    for (int i = 0; i < size; ++i)
    {
        if (arr[i] == value)
        {
            return i;
        }
    }
    return -1;
}
```

Step	Statement	Cost	Time
1	int i = 0;	c_1	1
2	i < size;	c_2	n + 1
3	arr[i] == value	c_3	n
4	return i;	c_4	0
5	++i	c_5	n
6	return -1;	c_6	1

$$t(n) = c_1 + c_2(n + 1) + c_3(n) + c_5(n) + c_6$$

$$= (c_2 + c_3 + c_5)n + (c_1 + c_2 + c_6)$$

$$= 3n + 3$$

Big-O = $O(n)$

LinkedList Methods

All costs can be considered 1 and n is the number of nodes in the LinkedList object.

```
void LinkedList::insert(T value)
{
     Node<T>* newNode = new Node<T>(value);
     if (head == nullptr)
     {
          head = newNode;
     }
     else
     {
          Node<T>* current = head;
          while (current->next != nullptr)
          {
               current = current->next;
          }
          current->next = newNode;
     }
     size++;
}
```

Step	Statement	Cost	Time
1	Node<T>* newNode = new Node<T>(value);	c_1	1
2	head == nullptr	c_2	1
3	head = newNode;	c_3	0
4	Node<T>* current = head;	c_4	1
5	current->next != nullptr	c_5	n
6	current = current->next;	c_6	n – 1
7	current->next = newNode;	c_7	1
8	size++;	c_8	1

$$T(n) = c_1 + c_2 + c_4 + c_5(n) + c_6(n - 1) + c_7 + c_8$$

$$= (c_5 + c_6)n + (c_1 + c_2 + c_4 - c_6 + c_7 + c_8)$$

$$= 2n + 4$$

Big-O = O(n)

```
void LinkedList::insertFront(T value)
{
	Node<T>* newNode = new Node<T>(value);
	newNode->next = head;
	head = newNode;
	size++;
}
```

Step	Statement	Cost	Time
1	`Node<T>* newNode = new Node<T>(value);`	c_1	1
2	`newNode->next = head;`	c_2	1
3	`head = newNode;`	c_3	1
4	`size++;`	c_4	1

$T(n) = c_1 + c_2 + c_3 + c_4$

$= 4$

Big-O = O(1)

```
T LinkedList::pop()
{
    if (head == nullptr)
    {
        throw runtime_error("List is empty");
    }
    Node<T>* current = head;
    if (current->next == nullptr)
    {
        T value = current->data;
        delete current;
        head = nullptr;
        size--;
        return value;
    }
    while (current->next->next != nullptr)
    {
        current = current->next;
    }
    T value = current->next->data;
    delete current->next;
    current->next = nullptr;
    size--;
    return value;
}
```

Step	Statement	Cost	Time
1	head == nullptr	c_1	1
2	throw runtime_error("List is empty");	c_2	0
3	Node<T>* current = head;	c_3	1
4	current->next == nullptr	c_4	1
5	T value = current->data;	c_5	0
6	delete current;	c_6	0
7	head = nullptr;	c_7	0
8	size--;	c_8	0
9	return value;	c_9	0
10	current->next->next != nullptr	c_{10}	n – 1
11	current = current->next;	c_{11}	n – 2
12	T value = current->next->data;	c_{12}	1
13	delete current->next;	c_{13}	1
14	current->next = nullptr;	c_{14}	1
15	size--;	c_{15}	1
16	return value;	c_{16}	1

$$
\begin{aligned}
T(n) &= c_1 + c_3 + c_4 + c_{10}(n - 1) + c_{11}(n - 2) + c_{12} + c_{13} + c_{14} + c_{15} + c_{16} \\
&= (c_{10} + c_{11})n + (c_1 + c_3 + c_4 - c_{10} - c_{11} - c_{11} + c_{12} + c_{13} + c_{14} + c_{15} + c_{16}) \\
&= 2n + 5
\end{aligned}
$$

Big-O = O(n)

```
T LinkedList::popFront()
{
    if (head == nullptr)
    {
        throw runtime_error("List is empty");
    }
    Node<T>* temp = head;
    T value = head->data;
    head = head->next;
    delete temp;
    size--;
    return value;
}
```

Step	Statement	Cost	Time
1	head == nullptr	c_1	1
2	throw runtime_error("List is empty");	c_2	0
3	Node<T>* temp = head;	c_3	1
4	T value = head->data;	c_4	1
5	head = head->next;	c_5	1
6	delete temp;	c_6	1
7	size--;	c_7	1
8	return value;	c_8	1

$T(n) = c_1 + c_3 + c_4 + c_5 + c_6 + c_7 + c_8$

$= 7$

Big-O = O(1)

```
bool LinkedList::remove(T value)
{
    if (head == nullptr)
    {
        return false;
    }

    if (head->data == value)
    {
        Node<T>* temp = head;
        head = head->next;
        delete temp;
        size--;
        return true;
    }

    Node<T>* current = head;

    while (current->next != nullptr && current->next->data != value)
    {
        current = current->next;
    }

    if (current->next == nullptr)
    {
        return false;
    }

    Node<T>* temp = current->next;
    current->next = current->next->next;
    delete temp;
    size--;
    return true;
}
```

Step	Statement	Cost	Time
1	head == nullptr	c_1	1
2	return false;	c_2	0
3	head->data == value	c_3	1
4	Node<T>* temp = head;	c_4	0
5	head = head->next;	c_5	0
6	delete temp;	c_6	0
7	size--;	c_7	0
8	return true;	c_8	0
9	Node<T>* current = head;	c_9	1
10	while (current->next != nullptr && current->next->data != value)	c_{10}	n
11	current = current->next;	c_{11}	n − 1
12	current->next == nullptr	c_{12}	1
13	return false;	c_{13}	0
14	Node<T>* temp = current->next;	c_{14}	1
15	current->next = current->next->next;	c_{15}	1
16	delete temp;	c_{16}	1
17	size--;	c_{17}	1
18	return true;	c_{18}	1

$$T(n) = c_1 + c_3 + c_9 + c_{10}(n) + c_{11}(n - 1) + c_{12} + c_{14} + c_{15} + c_{16} + c_{17} + c_{18}$$

$$= (c_{10} + c_{11})n + (c_1 + c_3 + c_9 - c_{11} + c_{12} + c_{14} + c_{15} + c_{16} + c_{17} + c_{18})$$

$$= 2n + 8$$

Big−O = O(n)

```
bool LinkedList::contains(T value) const
{
    Node<T>* current = head;
    while (current != nullptr)
    {
        if (current->data == value)
        {
            return true;
        }
        current = current->next;
    }
    return false;
}
```

Step	Statement	Cost	Time
1	Node<T>* current = head;	c_1	1
2	current != nullptr	c_2	n + 1
3	current->data == value	c_3	n
4	return true;	c_4	0
5	current = current->next;	c_5	n
6	return false;	c_6	1

$$T(n) = c_1 + c_2(n + 1) + c_3(n) + c_5(n) + c_6$$

$$= (c_2 + c_3 + c_5)n + (c_1 + c_2 + c_6)$$

$$= 3n + 3$$

Big-O = O(n)

```
size_t LinkedList::getSize() const
{
    return size;
}
```

Step	Statement	Cost	Time
1	`return size;`	c_1	1

T(n) = c_1

= 1

Big-O = O(1)

```
string LinkedList::toString() const override
{
    stringstream out;
    out << "[";
    Node<T>* current = head;
    while (current != nullptr)
    {
        out << current->data;
        if (current->next != nullptr)
        {
            out << ",";
        }
        current = current->next;
    }
    out << "]";
    return out.str();
}
```

Step	Statement	Cost	Time
1	`stringstream out;`	c_1	1
2	`out << "[";`	c_2	1
3	`Node<T>* current = head;`	c_3	1
4	`current != nullptr`	c_4	n + 1
5	`current->next != nullptr`	c_5	n
6	`out << ",";`	c_6	n
7	`current = current->next;`	c_7	n – 1
8	`out << "]";`	c_8	n
9	`return out.str();`	c_9	1
10	`stringstream out;`	c_{10}	1

$$T(n) = c1 + c2 + c3 + c4(n + 1) + c5(n) + c6(n) + c7(n - 1)$$

$$+ c8(n) + c9 + c10$$

$$= (c4 + c5 + c6 + c7 + c8)n + (c1 + c2 + c3 + c4 - c7 + c9 + c10)$$

$$= 5n + 5$$

Big-O = O(n)

```
void LinkedList::reverse()
{
    Node<T>* prev = nullptr;
    Node<T>* current = head;
    Node<T>* next = nullptr;
    while (current != nullptr)
    {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    head = prev;
}
```

Step	Statement	Cost	Time
1	`Node<T>* prev = nullptr;`	c_1	1
2	`Node<T>* current = head;`	c_2	1
3	`Node<T>* next = nullptr;`	c_3	1
4	`current != nullptr`	c_4	n + 1
5	`next = current->next;`	c_5	n
6	`current->next = prev;`	c_6	n
7	`prev = current;`	c_7	n
8	`current = next;`	c_8	n
9	`head = prev;`	c_9	1

$$T(n) = c_1 + c_2 + c_3 + c_4(n + 1) + c_5(n) + c_6(n) + c_7(n) + c_8(n) + c_9$$

$$= (c_4 + c_5 + c_6 + c_7 + c_8)n + (c_1 + c_2 + c_3 + c_4 + c_9)$$

$$= 5n + 5$$

Big-O = O(n)