Rank these 6 order notation formula from fastest(1) to slowest(6).

Solution: 1 $O(8 \cdot s \cdot w \cdot h)$
Solution: 4 $O((s \cdot w \cdot h)^8)$
Solution: 6 $O((8 \cdot w \cdot h)^s)$
Solution: 5 $O(w \cdot h \cdot 8^s)$
Solution: 2 or 3 $O((s + w \cdot h)^8)$
Solution: 2 or 3 $O(w \cdot h \cdot s^8)$

NOTE: The ordering of the '2' vs. '3' depends on the relative size of the variables h, w, and s.
If $w = h = s$: $(w + w \cdot w)^8 = w^{16} > w \cdot w \cdot w^8 = w^{10}$.
If $w = h$ & $s = w^2$: $(w^2 + w \cdot w)^8 = w^{16} < w \cdot w \cdot (w^2)^8 = w^{18}$.

Complete the function below named **reverse** that takes in an STL `list` as its only argument and returns an STL `vector` that contains the same list except in reverse order. You should use a *reverse iterator* and you may not use `push_back`.

Solution:

```
template <class T>
std::vector<T> reverse(const std::list<T> &my_list) {
  std::vector<T> answer (my_list.size());
  int i = 0;
  typename std::list<T>::const_reverse_iterator itr = my_list.rbegin();
  while (itr != my_list.rend()) {
    answer[i] = *itr;
    i++;
    itr++;
  }
  return answer;
}
```

You may not use STL `sort`. You may assume the input list does not contain any duplicates. And after calling the `FlipWords` function the list should not contain any duplicates.

Solution:

```
std::string reverse(std::string &word) {
  std::string answer(word.size(),' ');
  for (int i = 0; i < word.size(); i++) { answer[i] = word[word.size()-1-i]; }
  return answer;
}
void FlipWords(std::list<std::string> &words) {
  std::list<std::string>::iterator current = words.begin();
  while (current!= words.end()) {
    std::string flip = reverse(*current);
    if (flip == *current) {
      current = words.erase(current);
    } else {
      std::list<std::string>::iterator tmp = words.begin();
      while (tmp != words.end() && flip > *tmp) {
        tmp++;
      }
      if (tmp == words.end() || flip != *tmp) {
        words.insert(tmp,flip);
      }
      current++;
    }
  }
}
```


Recursion: examine frames of the stack

Dr. Memory for leaks (takes a long time to run

Backtrace to find crash

Gdb variable values

Watchpoint to see semantic errors.

List<int>::const_iterator it1 = lst.begin()

List<int>::iterator it2 = lst.begin()

Iterator operations:

advance	Advance iterator (function template)
distance	Return distance between iterators (function template)
begin C++11	Iterator to beginning (function template)
end C++11	Iterator to end (function template)
prev C++11	Get iterator to previous element (function template)
next C++11	Get iterator to next element (function template)

Iterators:

begin	Return iterator to beginning (public member function)
end	Return iterator to end (public member function)
rbegin	Return reverse iterator to reverse beginning (public member function)
rend	Return reverse iterator to reverse end (public member function)
cbegin C++11	Return const_iterator to beginning (public member function)
cend C++11	Return const_iterator to end (public member function)
crbegin C++11	Return const_reverse_iterator to reverse beginning (public member function)
crend C++11	Return const_reverse_iterator to reverse end (public member function)

Capacity:

empty	Test whether container is empty (public member function)
size	Return size (public member function)
max_size	Return maximum size (public member function)

Element access:

front	Access first element (public member function)
back	Access last element (public member function)

Now, write a second function. This **void** function will be passed a reference to an STL list of ints. In this function each zero in the list should be replaced with the sum of the adjacent numbers. A zero in the first or last position of the list should not be replaced. For example, if the list originally contained 1 0 11 16 0 0 50 75 85 90 0, the returned list will contain 1 12 11 16 16 66 50 75 85 90 0. Iterate through the list from left to right and replace the elements sequentially. Notice how consecutive zeros are handled. The first zero is replaced and the replacement value becomes the adjacent value for the next zero. That is, a list containing x 0 0 0 y will become x x x x+y y, where x and y are integers.

The zeros are to be replaced in the original list. Do not make a copy of the list. Iterate through the list and replace the elements. Do not use *std::replace* or *std::find*.

Solution:

```
void replace_zeros(std::list<int>& lst) {
  if (lst.size() == 0)
    return;

  std::list<int>::iterator it1 = lst.begin();
  it1++;

  while (it1 != lst.end())  {
    std::list<int>::iterator it_prev = it1;
    it_prev--;
    std::list<int>::iterator it_next = it1;
    it_next++;
    if (*it1 == 0 && it_next != lst.end()) {
```



```
      it1 = lst.erase(it1);
      it1 = lst.insert(it1, *it_prev + *it_next);
    }
    it1++;
  }
}
```

Memory Errors:

New used with [], then delete with []

** with new have to delete [] and delete

6.2 Stairs Constructor [/ 9]

Now write the constructor, as it would appear outside of the class declaration (because the implementation is > 1 line of code).

Solution:

```
template <class T> Stairs<T>::Stairs(int s, const T& val) {
  size = s;
  data = new T*[s];
  for (int i = 0; i < s; i++) {
    data[i] = new T[i+1];
    for (int j = 0; j <= i; j++) {
      data[i][j] = val;
    }
  }
}
```

6.3 Stairs Destructor [/ 5]

Now write the destructor, as it would appear outside of the class declaration (because the implementation is > 1 line of code).

Solution:

```
template <class T> Stairs<T>::~Stairs() {
  for (int i = 0; i < size; i++) {
    delete [] data[i];
  }
  delete [] data;
}
```

Above is outside of the class declaration so need :: stuff

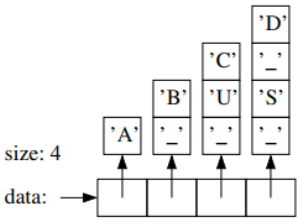
Changing a 2D data structure to a 1D data structure

Solution:

```
template <class T>
T* Flatten(T** data_2D, int m, int n){
  if(m <= 0 || n <= 0){
    return NULL;
  }

  T* ret = new T[m*n];
  for (int i=0; i<m; i++){
    for (int j=0; j<n; j++){
      ret[(i*n) + j] = data_2D[i][j];
    }
  }
  return ret;
}
```

```
Stairs<char> s(4,'_');
s.set(0,0,'A');
s.set(1,1,'B');
s.set(2,2,'C');
s.set(3,3,'D');
s.set(2,1,'U');
s.set(3,1,'S');
```



6.1 Stairs Class Declaration [/ 14]

First, fill in the blanks in the class declaration:

Solution:

```
template <class T>  class Stairs {

  public:
    // constructor

    Stairs(int s, const T& val);

    // destructor

    ~Stairs();

    // prototypes of 2 other important functions related to the constructor & destructor

```



```
    Stairs(const Stairs& s);
    Stairs& operator=(const Stairs& s);

    // modifier
    void set(int i, int j, const T& val) { data[i][j] = val; }

    /* NOTE: other Stair functions omitted */
  private:
    // representation

    int size;
    T** data;

};
```