ordered_insert

```
template<typename T, typename V>
typename std::vector<T>::iterator
ordered_insert(std::vector<T>& container,
               typename std::vector<T>::value_type v)
{
   std::vector<T>::iterator i(container.begin());
   while (i != container.end() && !(v < *i)) ++i;
   return container.insert(i, v);
}
```

ordered_overwrite

```
template<typename T, typename V>
typename std::vector<T>::iterator
ordered_overwrite(std::vector<T>& container,
                  typename std::vector<T>::value_type v)
{
    std::vector<T>::iterator i(container.begin());
    while (i != container.end() && *i < v) ++i;

    if (i == container.end() || v < *i)
        return container.insert(i, v);
    else
    {
        *i = v;
        return i;
    }
}
```

unique_append

```
template<typename T>
void unique_append(std::vector<T>& container,
                   typename std::vector<T>::value_type v)
{
    for (auto i(container.begin()); i != container.end(); ++i)
        if (value == *i) return;
    container.push_back(v);
}
```

Random Access Container Functions

```
template<typename CONTAINER>
inline void insert_at(CONTAINER& cont,
                      typename CONTAINER::size_type index,
                      typename CONTAINER::value_type value);

template<typename CONTAINER>
inline void erase_at(CONTAINER& cont,
                     typename CONTAINER::size_type index);

template<typename CONTAINER>
inline void swap_at(CONTAINER& cont,
                    typename CONTAINER::size_type a,
                    typename CONTAINER::size_type b);
```