Алгоритмы, не модифицирующие последовательность

```
Fun for_each(InIt first, InIt last, Fun f);
InIt find(InIt first, InIt last, const T& val);
InIt find_if(InIt first, InIt last, Pred pr);
InIt find_if_not(InIt first, InIt last, Pred pr);
FwdIt1 find_first_of(FwdIt1 first1, FwdIt1 last1, FwdIt2 first2, FwdIt2 last2);
FwdIt1 find_first_of(FwdIt1 first1,FwdIt1 last1,FwdIt2 first2,FwdIt2 last2,BinPred pr);
FwdIt1 find_end(FwdIt1 first1, FwdIt1 last1, FwdIt2 first2, FwdIt2 last2);
FwdIt1 find_end(FwdIt1 first1, FwdIt1 last1, FwdIt2 first2, FwdIt2 last2, BinPred pr);
FwdIt adjacent_find(FwdIt first, FwdIt last);
FwdIt adjacent_find(FwdIt first, FwdIt last, BinPred pr);
bool all_of(InIt first, InIt last, Pred pr);
bool any_of(InIt first, InIt last, Pred pr);
bool none_of(InIt first, InIt last, Pred pr);
size_t count(InIt first, InIt last, const T& val);
size_t count_if(InIt first, InIt last, Pred pr);
pair<InIt1, InIt2> mismatch(InIt1 first, InIt1 last, InIt2 x);
pair<InIt1, InIt2> mismatch(InIt1 first, InIt1 last, InIt2 x, BinPred pr);
bool equal(InIt1 first, InIt1 last, InIt2 x);
bool equal(InIt1 first, InIt1 last, InIt2 x, BinPred pr);
FwdIt1 search(FwdIt1 first1, FwdIt1 last1, FwdIt2 first2, FwdIt2 last2); // 2 ищется в 1
FwdIt1 search(FwdIt1 first1, FwdIt1 last1, FwdIt2 first2, FwdIt2 last2, BinPred eq);
FwdIt search_n(FwdIt first, FwdIt last, Size n, T val);//n подряд элементов val- 1-е вхожд-е
FwdIt search_n(FwdIt first, FwdIt last, Size n, T val, BinPred eq);
```

Алгоритмы, модифицирующие последовательность

```
OutIt copy(InIt first, InIt last, OutIt x);
OutIt copy_n(InIt first, Size n, OutIt result);
OutIt copy_if(InIt first, InIt last, OutIt x, Pred p);
BidIt2 copy_backward(BidIt1 first, BidIt1 last, BidIt2 x);
OutIt move(InIt first, InIt last, OutIt x); // вместо = используется move
FwdIt2 swap_ranges(FwdIt1 first, FwdIt1 last, FwdIt2 x); // диапазоны не должны перекрываться
void iter_swap(FwdIt1 x, FwdIt2 y);
OutIt transform(InIt first, InIt last, OutIt x, Unop uop);
OutIt transform(InIt1 first1, InIt1 last1, InIt2 first2, OutIt x, Binop bop);
void replace(FwdIt first, FwdIt last, const T& vold, const T& vnew);
void replace_if(FwdIt first, FwdIt last, Pred pr, const T& val);
OutIt replace_copy(InIt first, InIt last, OutIt x, const T& vold, const T& vnew);
OutIt replace_copy_if(InIt first, InIt last, OutIt x, Pred pr, const T& val);
void fill(FwdIt first, FwdIt last, const T& x);
void fill_n(OutIt first, Size n, const T& x);
void generate(FwdIt first, FwdIt last, Gen g);
void generate_n(OutIt first, Dist n, Gen g);
FwdIt remove(FwdIt first, FwdIt last, const T& val);
FwdIt remove_if(FwdIt first, FwdIt last, Pred pr);
OutIt remove_copy(InIt first, InIt last, OutIt x, const T& val);
OutIt remove_copy_if(InIt first, InIt last, OutIt x, Pred pr);
FwdIt unique(FwdIt first, FwdIt last);
FwdIt unique(FwdIt first, FwdIt last, BinPred pr);
OutIt unique_copy(InIt first, InIt last, OutIt x);
OutIt unique_copy(InIt first, InIt last, OutIt x, BinPred pr);
void reverse(BidIt first, BidIt last);
OutIt reverse_copy(BidIt first, BidIt last, OutIt x);
void rotate(FwdIt first, FwdIt middle, FwdIt last);
OutIt rotate_copy(FwdIt first, FwdIt middle, FwdIt last, OutIt x);
void random_shuffle(RanIt first, RanIt last);
void random_shuffle(RanIt first, RanIt last, Fun& f);
```

Алгоритмы разделения, сортировки и бинарного поиска последовательности

```
BidIt partition(BidIt first, BidIt last, Pred pr);
FwdIt stable_partition(FwdIt first, FwdIt last, Pred pr);
bool is_partitioned(InIt first, InIt last, Pred pr);
void sort(RanIt first, RanIt last);
void sort(RanIt first, RanIt last, Compare comp);
bool is_sorted(ForwIt first, ForwIt last);
bool is_sorted(ForwIt first, ForwIt last, Compare comp);
```

```
FwdIt is_sorted_until(FwdIt first, FwdIt last);
FwdIt is_sorted_until(FwdIt first, FwdIt last, Compare comp);
void stable_sort(BidIt first, BidIt last);
void stable_sort(BidIt first, BidIt last, Compare comp); Делается максимум N(logN)^2 сравнений
void partial_sort(RanIt first, RanIt middle, RanIt last); // до middle эл-ты отсортированы
void partial_sort(RanIt first, RanIt middle, RanIt last, Compare comp);
RanIt partial_sort_copy(InIt first1, InIt last1, RanIt first2, RanIt last2);
RanIt partial_sort_copy(InIt first1, InIt last1, RanIt first2, RanIt last2, Pred pr);
void nth_element(RanIt first, RanIt nth, RanIt last); все левее nth <= nth <= всех правее nth
void nth_element(RanIt first, RanIt nth, RanIt last, Compare comp);
FwdIt lower_bound(FwdIt first, FwdIt last, const T& val);
FwdIt lower_bound(FwdIt first, FwdIt last, const T& val, Compare comp);
FwdIt upper_bound(FwdIt first, FwdIt last, const T& val);
FwdIt upper_bound(FwdIt first, FwdIt last, const T& val, Compare comp);
pair<FwdIt, FwdIt> equal_range(FwdIt first, FwdIt last, const T& val);
pair<FwdIt, FwdIt> equal_range(FwdIt first, FwdIt last, const T& val, Compare comp);
bool binary_search(FwdIt first, FwdIt last, const T& val);
bool binary_search(FwdIt first, FwdIt last, const T& val, Compare comp);
OutIt merge(InIt1 first1, InIt1 last1, InIt2 first2, InIt2 last2, OutIt x);
OutIt merge(InIt1 first1, InIt1 last1, InIt2 first2, InIt2 last2, OutIt x, Compare comp);
void inplace_merge(BidIt first, BidIt middle, BidIt last);
void inplace_merge(BidIt first, BidIt middle, BidIt last, Compare comp);
```

Алгоритмы для работы с множествами

```
bool includes(InIt1 first1, InIt1 last1, InIt2 first2, InIt2 last2);
bool includes(InIt1 first1, InIt1 last1, InIt2 first2, InIt2 last2, Compare comp);
OutIt set_union(InIt1 first1, InIt1 last1, InIt2 first2, InIt2 last2, OutIt x);
OutIt set_union(InIt1 first1, InIt1 last1,InIt2 first2, InIt2 last2, OutIt x, Compare cmp);
OutIt set_intersection(InIt1 first1, InIt1 last1,InIt2 first2, InIt2 last2, OutIt x);
OutIt set_intersection(InIt1 first1, InIt1 last1,
        InIt2 first2, InIt2 last2, OutIt x, Compare cmp);
OutIt set_difference(InIt1 first1, InIt1 last1, InIt2 first2, InIt2 last2, OutIt x);
OutIt set_difference(InIt1 first1, InIt1 last1,
        InIt2 first2, InIt2 last2, OutIt x, Compare comp);
OutIt set_symmetric_difference(InIt1 first1, InIt1 last1,
        InIt2 first2, InIt2 last2, OutIt x);
OutIt set_symmetric_difference(InIt1 first1, InIt1 last1,
        InIt2 first2, InIt2 last2, OutIt x, Compare comp);
```

Минимумы и максимумы

```
const T& max(const T& x, const T& y);
const T& max(const T& x, const T& y, Compare cmp);
T max (initializer_list<T> il)
T max (initializer_list<T> il, Compare cmp)
const T& min(const T& x, const T& y);
const T& min(const T& x, const T& y, Compare cmp);
T min (initializer_list<T> il)
T min (initializer_list<T> il, Compare cmp)
FwdIt max_element(FwdIt first, FwdIt last);
FwdIt max_element(FwdIt first, FwdIt last, Compare cmp);
FwdIt min_element(FwdIt first, FwdIt last);
FwdIt min_element(FwdIt first, FwdIt last, Compare cmp);
pair<const T&,const T&> minmax(const T& x, const T& y);
pair<const T&,const T&> minmax(const T& x, const T& y, Compare cmp);
pair<T,T> minmax(initializer_list<T> il);
pair<T,T> minmax(initializer_list<T> il, Compare cmp);
bool lexicographical_compare(InIt1 first1, InIt1 last1, InIt2 first2, InIt2 last2);
bool lexicographical_compare(InIt1 first1, InIt1 last1, InIt2 first2, InIt2 last2, Compare
cmp);
```

Перестановки

```
bool next_permutation(BidIt first, BidIt last);
bool next_permutation(BidIt first, BidIt last, Compare cmp);
bool prev_permutation(BidIt first, BidIt last);
bool prev_permutation(BidIt first, BidIt last, Compare cmp);
bool is_permutation(InIt1 first1, InIt1 last1, InIt2 first2);
```