

## \* Standard Function Classes:

→ Genelleştirilmiş, birden operasyonların tembleleri

```
#include <functional>
// bu header altında

int main()
{
    using namespace std;

    auto x1 = plus<int>{}(3, 5);
    auto x2 = minus<int>{}(3, 5);
    auto x3 = multiplies<int>{}(3, 5);
    auto x4 = divides<int>{}(3, 5);
    auto x5 = modulus<int>{}(3, 5);
    auto x6 = negate<int>{}(3);
    auto x7 = less<int>{}(3, 5);
    auto x8 = greater<int>{}(3, 5);
    auto x9 = less_equal<int>{}(3, 5);
    auto x10 = greater_equal<int>{}(3, 5);
    auto x11 = equal_to<int>{}(3, 5);
    auto x12 = not_equal_to<int>{}(3, 5);
    auto x13 = logical_and<int>{}(3, 5);
    auto x14 = logical_or<int>{}(3, 5);
    auto x15 = logical_not<int>{}(3);
    auto x16 = bit_and<int>{}(3, 5);
    auto x17 = bit_or<int>{}(3, 5);
    auto x18 = bit_xor<int>{}(3, 5);
    auto x19 = bit_not<int>{}(3);
```

## \* Stable / Unstable Sort:

→ Stable sort, sıralamaya konan değerler, sıralanmadan önceki aynı değere sahip olur. Aynı değere sahipse, aynı yerde kalır.

```
using Person = std::pair<int, std::string>;
```

```
int main()
{
    constexpr int size{ 10'000 };
    using namespace std;

    vector<Person> pvec;
    pvec.reserve(size);
    const auto f = [] { return pair{ Irand{0, 100}(), rname() }; };
    generate_n(back_inserter(pvec), size, f);

    ofstream ofs{ "out.txt" };
    if (!ofs) {
        cerr << "out.txt dosyasi olusturulamadi\n";
        exit(EXIT_FAILURE);
    }

    for (const auto& p : pvec) {
        ofs << p.first << ' ' << p.second << '\n';
    }
}
```

```

int main()
{
    constexpr int size{ 10'000 };
    using namespace std;

    vector<Person> pvec;
    pvec.reserve(size);
    const auto f = [] {return pair{ Irand{0, 999}(), rname() }; };
    generate_n(back_inserter(pvec), size, f);

    sort(pvec.begin(), pvec.end(), [](const Person& p1, const Person& p2)
    {
        return p1.first < p2.first;
    });

    stable_sort(pvec.begin(), pvec.end(), [](const Person& p1, const Person& p2)
    {
        return p1.second < p2.second;
    });
}

ofstream ofs{ "out.txt" };
if (!ofs) {
    cerr << "out.txt dosyasi olusturulamadi\n";
    exit(EXIT_FAILURE);
}

ofs << left;
for (const auto&[id, name] : pvec) {
    ofs << setw(10) << id << ' ' << name << '\n';
}

```

No issues found

	main.cpp
57	abdi
69	abdi
87	abdi
91	abdi
148	abdi
168	abdi
177	abdi
195	abdi
227	abdi
275	abdi
283	abdi
324	abdi
331	abdi
372	abdi
396	abdi
406	abdi
418	abdi
438	abdi
439	abdi
462	abdi
464	abdi
484	abdi
520	abdi
522	abdi
557	abdi
573	abdi
579	abdi
610	abdi
629	abdi
762	abdi
775	abdi
789	abdi
792	abdi
846	abdi

→ int degerine gore ve  
alfabetik siraya gore stable  
sortlandı

## \*Partial Sort

```
using Person = std::pair<int, std::string>;  
  
int main()  
{  
    constexpr int size{ 10'000 };  
    using namespace std;  
  
    vector<Person> pvec;  
    pvec.reserve(size);  
    const auto f = [] { return pair{ Irand{0, 10000}(), rname() }; };  
    generate_n(back_inserter(pvec), size, f);  
  
    partial_sort(pvec.begin(), pvec.begin() + 20, pvec.end(), [](const Person& p1, const Person& p2)  
    {  
        return p1.first > p2.first;  
    });  
    J similar righten legeze  
    yapınca  
    NL 20  
    önce sıralanı  
  
    ofstream ofs{ "out.txt" };  
    if (!ofs) {  
        cerr << "out.txt dosyası oluşturulamadı\n";  
        exit(EXIT_FAILURE);  
    }  
  
    ofs << left;  
    for (const auto&[id, name] : pvec) {  
        ofs << setw(10) << id << ' ' << name << '\n';  
    }  
}
```

baska bir binary predicate olursa less işe girebilir

bu lambda expression otomatik less işe girebilir

→ Farklı range arasındaki sort olması garantisiz yok.

## \*Partial Sort Copy:

→ Baska range'e partial sort eder.

## \*nth\_element:

→ Bir range'de N. konumda hangi tane olması gerekiyorsa onu yineleter. O taneın olması sırasız ama konumdan değil, o tanein sonrasi sırasız ama ondan büyük.

## \* Median hesaplamasında医疗保险:

```
int main()  
{  
    using namespace std;  
  
    vector<int> ivec;  
    ivec.reserve(1'000'000);  
  
    rfill(ivec, 100'000, Irand{ 0, 1'000'000 });  
    nth_element(ivec.begin(), ivec.begin() + ivec.size() / 2, ivec.end());  
    cout << "median = " << ivec[ivec.size() / 2] << "\n";  
    l+1  
    nth  
    last  
}
```

### \* Partition:

→ grupta göre kendi içinde sıralama. **BT** koşulu göre gruplenmesi. → Binary predicate isisi.

→ **Partition Point:** koşulu sağlayan ilk elemanın konumu. Eğer range'daki her tane koşulu sağlarsa, partition point = end() / sondan 1.

```
using namespace std;

vector<int> ivec;
ivec.reserve(100);

rfill(ivec, 100, Irand{ 0, 1000 });
print(ivec);

auto pp = partition(ivec.begin(), ivec.end(), [] (int x) {return x % 2 == 0; });
print(ivec);

if (pp != ivec.end()) {
    std::cout << "partition point : " << *pp << "\n"
```

```
Microsoft Visual Studio Debug Console
619 949 689 205 189 165 229 966 677 418 709 687 229 257 953 753 689 393 610 908 384 825 185 846 492 286 868 556 310 583 270 164 34
5 905 625 86 663 650 526 932 418 68 594 805 546 106 814 997 650 767 941 403 694 276 345 659 862 164 318 948 78 267 694 974 759 92
387 887 545 149 938 555 148 630 800 710 672 695 690 273 624 971 666 940 117 259 905 198 98 61 175 689 954 623 904 449 876 570 699
604
604 570 876 904 954 98 198 966 940 418 666 624 690 672 710 800 630 148 610 908 384 938 92 846 492 286 868 556 310 974 270 164 694
78 948 86 318 650 526 932 418 68 594 164 546 106 814 862 650 276 694 403 941 767 345 659 997 805 663 625 905 267 345 583 759 185 3
87 887 545 149 825 555 393 689 753 953 257 695 229 273 687 971 709 677 117 259 905 229 165 61 175 689 189 623 205 449 689 949 699
619
```

### \* Partition Copy:

→ koşulu sağlayanlar bir yere, sağlanmamasılar başka yere toplanır.

→ ilk tane aynı destination range başlangıç iteratori şerefleri.

partition point.

```
int main()
{
    using namespace std;

    vector<string> svec;
    svec.reserve(1000);

    rfill(svec, 1000, rname);

    vector<string> ok_vec(1000);
    vector<string> not_ok_vec(1000);

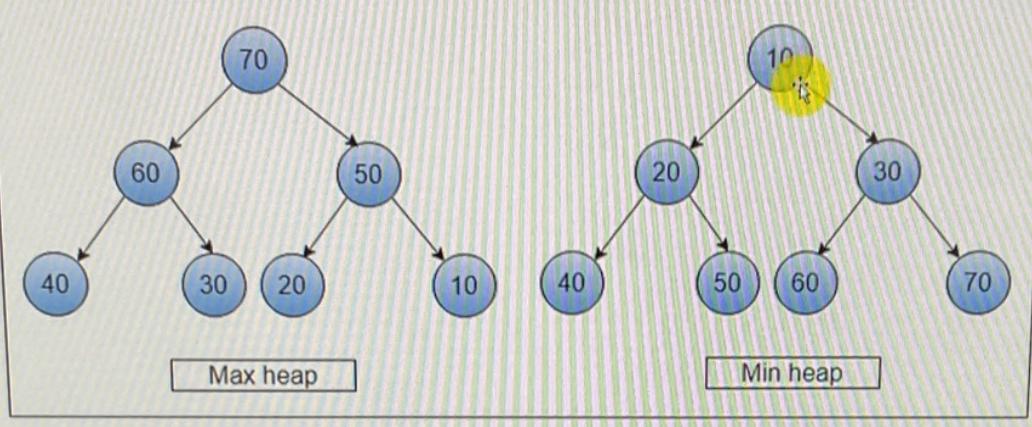
    auto ip = partition_copy(svec.begin(), svec.end(), ok_vec.begin(), back_inserter(not_ok_vec),
        [] (const std::string& s) {return s.contains('a'); });

    return 0;
}
```

eger vector tenimizdeysse  
eger vector tenimizdeysse  
return degeri pair.

Cop 20'de esitedi.

## Heap Algoritmaları:



→ Fotot heap sırt olmasızdır binary tree complete olmaz. (Soy bos olsalar, soyu boş olamaz)

→ Priority Queue'ca kullanılır.

→ Heap vektörel olarak ifade edilebilir.



→ Insertion =  $O(\log N)$

→ Vec yapsın heap yapın = heapify → STL klasörü = make\_heap()

→ make\_heap max 3<sup>rd</sup> distance  $\rightarrow O(N)$  kompleksite

→ std::greater ile min heap , std::less ile max heap.

```
using namespace std;
```

$\rightarrow O(\log N)$

```
vector<int> ivec;
rfill(ivec, 16, Irand(0, 100));

make_heap(ivec.begin(), ivec.end(), greater{});
print(ivec);
```

default oluyor max heap (lesser)
  
min heap

↳ sonda eklenen yaptığımızda, heap yapısı bozulur. Bu yüzden, önce sonda eklenen, sonra push\_heap yapılmalıdır.

```
int main()
{
    using namespace std;

    vector<int> ivec;
    rfill(ivec, 16, Irand(0, 100));

    make_heap(ivec.begin(), ivec.end());
    //print(ivec);
    ivec.push_back(53);
    push_heap(ivec.begin(), ivec.end());

    cout << is_heap(ivec.begin(), ivec.end()) << "\n";
    ↳ heap not
    degil mi one gore return true/false
```

$\rightarrow$  heap, en büyük değerle en hızlı erittirme iddiası.

```
int main()
{
    using namespace std;

    vector<string> svec;
    rfill(svec, 39, [] {return rname() + ' ' + rfname(); });

    make_heap(svec.begin(), svec.end());

    while (!svec.empty()) {
        pop_heap(svec.begin(), svec.end());
        std::cout << svec.back() << '\n';
        (void)getchar();
        svec.pop_back();
    }
}
```

\* Sort\_heap: + once make heap yapisi gerekir.

Sonra sort heap ile sortlansı.

\* is\_sorted:

```
13
14 int main()
15 {
16     using namespace std;
17
18     vector<int> ivec{ 345912, 123765, 2345, 981, 765, 45, 5, 3, -1};
19     cout << boolalpha;
20
21     cout << is_sorted(ivec.begin(), ivec.end(), greater{}) << "\n"; bogaldan because degisik arraye bakar.
22
23     ↓
24     true yada false dener.
25 }
```

\* is\_sorted\_until: → return degerit iterator.

\* Sırenin borulduğu konumun indexi

```
13
14 int main()
15 {
16     using namespace std;
17
18     vector<int> ivec{ 345912, 123765, 2345, 14, 981, 765, 45, 5, 3, -1}; 1. tigde son bosdu
19
20     auto std::vector<int>::iterator iter = is_sorted_until(_First:ivec.begin(), _Last:ivec.end(), _Pred:greater{}); I
21
22     cout << distance(_First:ivec.begin(), _Last:iter) << "\n"; → Gelen: 4
23
24
25 }
```

\* is\_partitioned:

→ partition edip edilmeyen gore true / false dener.

```
13
14 int main()
15 {
16     using namespace std;
17
18     vector<int> ivec{ 3, 7, 9, 1, 6, 4, 2}; +ticer      ciftler
19
20     cout << is_partitioned(ivec.begin(), ivec.end(), [](int x) {return x % 2 != 0; }) << "\n"; I tek olanlarla gore partition bolum
21
22     true / 1
23
24
25 }
```

\* partition\_point:

```
13
14 int main()
15 {
16     using namespace std;
17
18     vector<int> ivec{ 3, 7, 9, 1, 6, 4, 2}; I
19
20     auto iter = partition_point(ivec.begin(), ivec.end(), [](int x) {return x % 2 != 0; });
21     std::cout << "*iter = " << *iter << "\n"; → 6
22
23
24 }
```

\* max\_element:  $O(N)$  karmaşıklığında

→ return değeri, max elementin Olduğu konumun iteratörü

```
int main()
{
    using namespace std;

    vector<Date> dvec;

    rfill(dvec, 10, Date::random);
    print(dvec, "\n");
    auto max_iter = max_element(dvec.begin(), dvec.end());
    cout << "max = " << *max_iter << "\n";
    cout << "max idx = " << distance(dvec.begin(), max_iter) << "\n";
```

\* Aynı durum, min\_element için de olur.

```
int main()
{
    using namespace std;

    vector<Date> dvec;
    max_element(dvec.begin(), dvec.end())->year() // en büyük
    // yili ortadır.
    // Iteratörlerin pointer gibi → operator++
    cout << "max_idx = " << max_element(dvec.begin(), dvec.end()) << "\n";
```

\* for\_each():

The simplest example is the for\_each() algorithm, which calls a user-defined function for each element of the specified range. Consider the following example:

```
int main()
{
    using namespace std;

    vector<int> ivec{ 7,9,6,3,40,5,7,6,98,2,346,79,8, 53 };

    for_each(ivec.begin(), ivec.end(), [] (int& x) {
        x++;
    });

    for (int i : ivec)
        std::cout << i << " ";
}
```

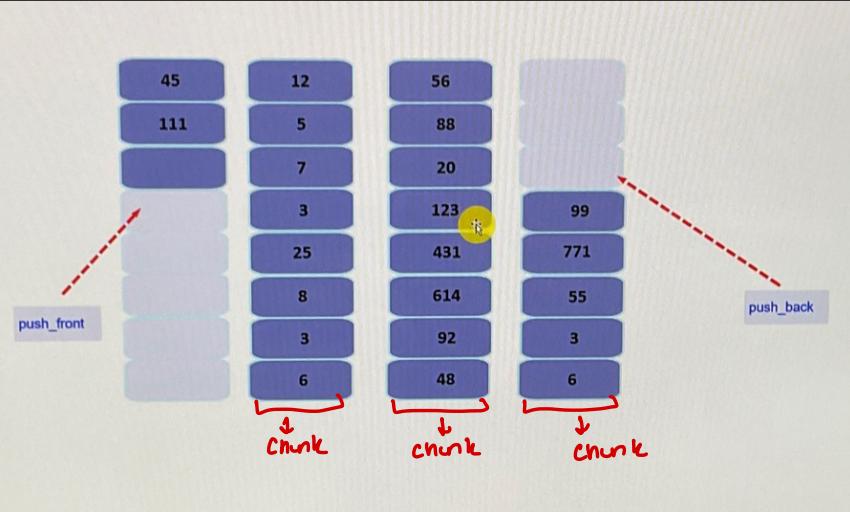
\* vector<bool>: → Bu'u STL kütüphanesinden oku. ve videoda öğrenile ~ 02:30:00  
→ vector bool ters mozi!

\* Deque Container:  $\rightarrow$  Double ended queue

\* Dinamik dizilerin altısı

\* Her bastırma eklemek, her sondan eklemek / silmek = Amortized Constant time  $\rightarrow$  Her bastırma, her sondan erişim genetik yerde

\* Aynıda indexler erişim mümkün  $\rightarrow O(1)$



$\rightarrow$  Deque, chunk chunk memory allende ekr.

$\leftarrow$  Contiguous memory allocation for chunks