

```
1 // range = [l, r), return last value causing "t" in evalfunc that returns l->[t,...,t,f,...,f)->r
2 // NOTE: if [f,...,f) then return l - 1, if [l, r) = empty set then invalid use
3 template<class val_t, class bsargv_t>
4 val_t lower_bsearch(val_t l, val_t r, const bsargv_t &v, bool (*evalfunc)(val_t, const bsargv_t&)) {
5     if (r - l == 1) {
6         if (evalfunc(l, v)) return l;
7         else return l - 1;
8     }
9     val_t m = (l + r) / 2;
10    if (evalfunc(m, v)) return lower_bsearch<val_t, bsargv_t>(m, r, v, evalfunc);
11    else return lower_bsearch<val_t, bsargv_t>(l, m, v, evalfunc);
12 }
13
14 // range = [l, r), return first value causing "t" in evalfunc that returns l->[f,...,f,t,...,t)->r
15 // NOTE: if [f,...,f) then return r, if [l, r) = empty set then invalid use
16 template<class val_t, class bsargv_t>
17 val_t upper_bsearch(val_t l, val_t r, const bsargv_t &v, bool (*evalfunc)(val_t, const bsargv_t&)) {
18     if (r - l == 1) {
19         if (evalfunc(l, v)) return l;
20         else return r;
21     }
22     val_t m = (l + r) / 2;
23     if (evalfunc(m, v)) return upper_bsearch<val_t, bsargv_t>(l, m, v, evalfunc);
24     else return upper_bsearch<val_t, bsargv_t>(m, r, v, evalfunc);
25 }
26
27 struct bsargv_t {
28     //
29 };
30
31 bool evalfunc(int val, const bsargv_t &v) {
32     //
33     return true;
34 }
```