```cpp
// range = [l, r), return last value causing "t" in evalfunc that returns l->[t,...,t,f,...,f)->r
// NOTE: if [f,...,f) then return l - 1, if [l, r) = empty set then invalid use
template<class val_t, class bsargv_t>
val_t lower_bsearch(val_t l, val_t r, const bsargv_t &v, bool(*evalfunc)(val_t, const bsargv_t&)) {
  if (r - l == 1) {
    if (evalfunc(l, v)) return l;
    else return l - 1;
  }
  val_t m = (l + r) / 2;
  if (evalfunc(m, v)) return lower_bsearch<val_t, bsargv_t>(m, r, v, evalfunc);
  else return lower_bsearch<val_t, bsargv_t>(l, m, v, evalfunc);
}

// range = [l, r), return first value causing "t" in evalfunc that returns l->[f,...,f,t,...,t)->r
// NOTE: if [f,...,f) then return r, if [l, r) = empty set then invalid use
template<class val_t, class bsargv_t>
val_t upper_bsearch(val_t l, val_t r, const bsargv_t &v, bool(*evalfunc)(val_t, const bsargv_t&)) {
  if (r - l == 1) {
    if (evalfunc(l, v)) return l;
    else return r;
  }
  val_t m = (l + r) / 2;
  if (evalfunc(m, v)) return upper_bsearch<val_t, bsargv_t>(l, m, v, evalfunc);
  else return upper_bsearch<val_t, bsargv_t>(m, r, v, evalfunc);
}

struct bsargv_t {
  //
};

bool evalfunc(int val, const bsargv_t &v) {
  //
  return true;
}
```