inline size\_t hyperfloor(unsigned long n)

{

return 1<<floor\_log2(n);

}

template<typename RandomIterator>

void build\_hp\_tree(RandomIterator begin\_in,

RandomIterator beyond\_in,

RandomIterator begin\_out,

RandomIterator beyond\_out,

int height,

int inc) {

int bottom\_height = ((height==2)?1:hyperfloor(height-1));

int top\_height = height-bottom\_height;

int bottom\_size = (1<<bottom\_height)-1;

int top\_size = (1<<top\_height)-1;

if (top\_height==1 && bottom\_height==1) {

begin\_out[1] = begin\_in[0];

begin\_out[0] = begin\_in[1\*inc];

begin\_out[2] = begin\_in[2\*inc];

return;

}

if (top\_height==1) {

begin\_out[0] = begin\_in[bottom\_size\*inc];

} else {

build\_hp\_tree(begin\_in+bottom\_size\*inc,

beyond\_in,

begin\_out,

beyond\_out,

top\_height,

bottom\_size\*inc+inc);

}

for(int i=0;i<=top\_size;i++) {

build\_hp\_tree(begin\_in+(i\*bottom\_size+i)\*inc,

beyond\_in+(i+1)\*bottom\_size\*inc+i,

begin\_out+top\_size+i\*bottom\_size,

beyond\_out,

bottom\_height,

inc);

}

}

16 8 4 12 2 1 3 6 5 7 10 9 11 14 13 15 24 20 28 18 17 19 22 21 23 26 25 27 30 29 31 32

template<typename element>

class precomputed\_table {

public:

precomputed\_table(element height) : D(height),

T(height),

B(height),

Pos(height) {

D.resize(height);

T.resize(height);

B.resize(height);

Pos.resize(height);

build\_precomputed\_table(height, 0);

}

void initialise() {

Pos[0] = 0;

};

vector<element> D;

vector<element> T;

vector<element> B;

vector<element> Pos;

private:

void build\_precomputed\_table(element height, element depth) {

element bottom\_height = ((height==2)?1:hyperfloor(height-1));

element top\_height = height-bottom\_height;

if (height==1) return;

D[depth+top\_height] = depth;

T[depth+top\_height] = (1<<top\_height)-1;

B[depth+top\_height] = (1<<bottom\_height)-1;

Pos[depth+top\_height] = 0;

build\_precomputed\_table(top\_height, depth);

build\_precomputed\_table(bottom\_height, depth+top\_height);

}

};