7.8

<a>最差的情况下，栈深度为n；

<b>先调用大划分的递归,再调用小划分的递归.栈深度变为log n；

7.9

2 6 4 0 1 3 5 7,共交换六次

7.16

<a>

int num[3];

void sort()

{

if (num[1] < num[0])

if (num[2] < num[0])

if (num[2] < num[1])

swap(num[0], num[2]);

else {

swap(num[0], num[1]);

swap(num[1], num[2]);

}

else

swap(num[0], num[1]);

else

if (num[2] < num[1])

if (num[2] < num[1]) {

swap(num[0], num[2]);

swap(num[1], num[2]);

}

else

swap(num[1], num[2]);

else

}

最佳情况需要2次比较,即无需重新排.最差情况需要3次比较.3个数共有6种情况,加在一起共16次,即平均次数为8/3次

<b>先利用三个数的比较法排好序,第四个数与中间作比较,小了和第一个数比,大了和第二个数比,需要两次比较.第五个数与4个数的第二个数比,小了就再比一次,大了可能需要比三次.所以最佳情况为2+2+2=6次,最差为3+2+3=8次,平均为8/3+2+12/5=7.567次.

<c>最佳12次,最差19次,平均14.435次

7.17

int n = 30000;

int A[n];

template <typename E, class getKey>

void binsort(E A[], int n) {

List<E> B[n];

E item;

for (int i = 0; i<n; i++) B[A[i]].append(getKey::key(A[i]));

for (int i = 0; i<n; i++)

for (B[i].setStart(); B[i].getValue(item); B[i].next())

output(item);

}

7.18

找中值最适合先排序,然后取中间值即可

<a>不断遍历比较,最差需要n

<b>同样需要n

<c>同样需要n

<d>使用快速排序,需要nlogn

<e>先排序nlogn再依次遍历,保存遇到的最多的数的次数,又需要 n,所以n+nlogn.

7.20

LList<int> mergesort(LList<int> inlist) {

LList<int> templist[2];

if (inlist.length() <= 1) return inlist;

inlist.setStart();

int curr = 0;

while (!inlist.isEmpty()) {

int item;

inlist.remove(item);

templist[curr].append(item);

curr = (curr + 1) % 2;

}

mergesort(templist[0]);

mergesort(templist[1]);

templist[0].setFirst();

templist[1].setFirst();

while (!templist[0].isEmpty() ||

!templist[1].isEmpty()) {

if (templist[0].isEmpty()) {

templist[1].remove(item);

inlist.append(item);

}

else if (templist[1].isEmpty()) {

templist[0].remove(item);

inlist.append(item);

}

else if (templist[0].currValue() <

templist[1].currValue()) {

item = templist[0].remove();

inlist.append(item);

}

else {

item = templist[1].remove();

inlist.append(item);

}

}

return inlist;

}

7.22

无论输入数据n有多少,所有基于比较的排序都可以用一棵树来模拟.树已有n个节点代表数据,而树的叶子节点最差情况即为最大层数logn,所以比较logn次可找到key对应的value.