Library Sort Project

I had code up the Library Sort, Insertion Sort and Quick Sort in C++ language. I generated 10 test cases in the CS583 folder to test my algorithm (name from data0~9). I used the algorithm in chapter 5 to generate random size permutation of random positive integers. All the cases are generated randomly in one program. I also included three algorithm output files of each of the ten test cases. I included a sorting clock in every sorting algorithm, which represent the times that entered a loop. In the test result, sort time is the real time in second that runs on my computer. I use the insertion-sort and quick-sort in textbook to compare library-sort. The test case shows that sometimes Library-Sort preforms better than Quick-Sort O(n log n) and always much more better than Insertion-Sort O(n2). But the problem is the expanded array is too large and too wasteful. I append the result of 10 cases and the draft design of my own.

The whole process of Library sort is different from insertion sort because this algorithm is using binary search. At first round, directly insert the first element to the new expanded array. Then the second round start with a rebalance process, which place all the elements in equal gaps. The way to calculate the gaps that I used is gaps = 2\*(1+EPSILON). Rebalance operation is quite simple and easy. The expanded array will be the twice of the previous array. Define *i* as the round of a library sort, the expanded array will increase to (1+EPSILON)\*2^*i*, which is twice of the previous array (1+EPSILON)\*2^(*i*+1). The rebalance process only deal with 2^(*i*) elements. The gaps can be calculated easily by (1+EPSILON)\*2^(*i*+1) / 2^(*i*). In this case, all the elements can be separated evenly. Moreover, the library sort runs rebalance only logN times, which greatly reduce the cost the high cost of insertion-sort.

void rebalance(int \* array, int range)

{

int i = range/2-1;

int new\_pos=range - (2+2\*EPSILON);

while (i >= 0 && new\_pos > i) {

lib\_clock\_count++;

if (array[i] != NIL) {

array[new\_pos] = array[i];

array[i] = NIL;

new\_pos -= (2+2\*EPSILON);

}

i--;

}

}

When the searching range is smaller than a gap, the best way to find the position is insertion sort. The binary search function will find the position of next elements. When the searching range is greater than 2\*(1+EPSILON), start with the binary search greatly reduce the cost of find position in insertion-sort. Until the range is less than 2\*(1+EPSILON), the cost of insertion-sort is less than quick-sort if EPSILON<4. This was demonstrated in textbook that insertion sort runs faster when data size equals to 10.

The insertion-sort was adopted inside small gaps. There are two situations. One is reach the boundary of the expanded array. The solution of this situation is move all the elements left. Another situation is there might be empty space inside small 2\*(1+EPSILON) gaps, since the left move operation break the consistence inside the small gaps. The solution of this situation is to check the consistency of the searching gap. If the last element of the small gap is not empty, the insert operation must go through all the gaps and must ignore the gaps.

Since the expanded array is always greater than the origin array. The last round directly skips the rebalance and expansion and inserts all elements into the array. There are two reasons for why I did this. Firstly, the cost of rebalance and expansion are very high at the following round. Secondly, if the size of expanded array changes into (1+EPSILON) times data size, it might destroy the binary property after rebalance operation at the last round. The number of supported elements in the last round is (1+EPSILON) \*2^*i*. No one can guarantee n / (1+EPSILON)\*2^*i* to be an even gap. If the gaps are not even, it is hard to locate the middle number of the expanded array.

Here is my out put data:

**Please Type the data set you want to test(Press 0~9 then Enter)**

**Please Type -1 to generate random permuation**

**0**

**File name:/Users/kang/Documents/MyOJProject/CS583/CS583/data0**

**Data size:16807**

**Library sort need a EPSILON:4**

**Library sort time:0.007662**

**Library sort clock:132291**

**Insertion sort time:0.227573**

**Insertion sort clock:70745318**

**Quick sort time:0.002007**

**Quick sort clock:318975**

**1**

**File name:/Users/kang/Documents/MyOJProject/CS583/CS583/data1**

**Data size:9235**

**Library sort need a EPSILON:4**

**Library sort time:0.004011**

**Library sort clock:68545**

**Insertion sort time:0.071709**

**Insertion sort clock:21237164**

**Quick sort time:0.000988**

**Quick sort clock:150428**

**2**

**File name:/Users/kang/Documents/MyOJProject/CS583/CS583/data2**

**Data size:11246**

**Library sort need a EPSILON:4**

**Library sort time:0.004759**

**Library sort clock:74638**

**Insertion sort time:0.104754**

**Insertion sort clock:31875551**

**Quick sort time:0.001247**

**Quick sort clock:187866**

**3**

**File name:/Users/kang/Documents/MyOJProject/CS583/CS583/data3**

**Data size:21469**

**Library sort need a EPSILON:4**

**Library sort time:0.009453**

**Library sort clock:146197**

**Insertion sort time:0.362158**

**Insertion sort clock:114827195**

**Quick sort time:0.003294**

**Quick sort clock:373105**

**4**

**File name:/Users/kang/Documents/MyOJProject/CS583/CS583/data4**

**Data size:41619**

**Library sort need a EPSILON:4**

**Library sort time:0.018609**

**Library sort clock:288677**

**Insertion sort time:1.40587**

**Insertion sort clock:433471140**

**Quick sort time:0.005269**

**Quick sort clock:780820**

**5**

**File name:/Users/kang/Documents/MyOJProject/CS583/CS583/data5**

**Data size:66531**

**Library sort need a EPSILON:4**

**Library sort time:0.028353**

**Library sort clock:527163**

**Insertion sort time:3.67349**

**Insertion sort clock:1108635832**

**Quick sort time:0.0101**

**Quick sort clock:1282622**

**6**

**File name:/Users/kang/Documents/MyOJProject/CS583/CS583/data6**

**Data size:91645**

**Library sort need a EPSILON:4**

**Library sort time:0.041673**

**Library sort clock:602485**

**Insertion sort time:6.64411**

**Insertion sort clock:2098960335**

**Quick sort time:0.011697**

**Quick sort clock:1831447**

**7**

**File name:/Users/kang/Documents/MyOJProject/CS583/CS583/data7**

**Data size:74229**

**Library sort need a EPSILON:4**

**Library sort time:0.032732**

**Library sort clock:550277**

**Insertion sort time:4.37489**

**Insertion sort clock:1383468213**

**Quick sort time:0.011222**

**Quick sort clock:1440149**

**8**

**File name:/Users/kang/Documents/MyOJProject/CS583/CS583/data8**

**Data size:39261**

**Library sort need a EPSILON:4**

**Library sort time:0.018885**

**Library sort clock:281543**

**Insertion sort time:1.20767**

**Insertion sort clock:385337288**

**Quick sort time:0.006568**

**Quick sort clock:794988**

**9**

**File name:/Users/kang/Documents/MyOJProject/CS583/CS583/data9**

**Data size:97154**

**Library sort need a EPSILON:4**

**Library sort time:0.042095**

**Library sort clock:619102**

**Insertion sort time:7.51233**

**Insertion sort clock:-1928991266**

**Quick sort time:0.013072**

**Quick sort clock:2023843**

