Bin packing heuristics

Group05

黄芯芯

吴蓉

冯焕

邵维

Date: 2009.1.15

Content

Chapter 1: Introduction

This project requires us to implement and compare the performance (both in time and number of bins used) of the various bin packing heuristics, including the on-line, next fit, first fit, best fit, and first-fit decreasing algorithms.

As we all know, in computational complexity theory, the bin packing problem is a combinatorial NP-hard problem. In it, objects of different volumes must be packed into a finite number of bins of capacity V in a way that minimizes the number of bins used.

On the purpose of this report, we should compile different programs to run those four algorithms.

Our results give a thorough show of the performance of the four algorithms. The first fit algorithm provides a fast but often nonoptimal solution, involving placing each item into the first bin in which it will fit. It requires Θ(N log N) time, where n is the number of elements to be packed. The first decreasing algorithm can be made much more effective by first sorting the list of elements into decreasing order. And for longer lists may increase the running time of the algorithm. Next-fit algorithm differs from first-fit algorithm in that a first-fit algorithm places each item into the first bin in which it will fit, whereas a next-fit algorithm just check whether a new item fits in the same bin as the last item and just

create a new bin if not, and is significantly faster than the first-fit. Of course, it uses more bins. Best fit is the algorithm places objects in the tightest spot among all bins. It perform better for random inputs and needs O(NlogN) time.

Chapter 2: Data Structure / Algorithm Specification

In this project, we use different data structures to implement the four bin packing algorithms, aiming to get lower time complexity. Here, we will introduce the data structures used in the four algorithms as well as how the algorithms work.

The meaning of the inputs in the pseudo codes is shown as follows:

Num: the number of items to packing in the bins;

Item[]: an array to store the weight of each item;

C: the contain of the bins.

Algorithm 1: Next Fit (on-line)

Definition: When processing any item, we check to see whether it fits in the same bin as the last item. If it does, it is placed there; otherwise, a new bin is created.

Data Structure: To implement the algorithm in linear time complexity is easy. Therefore, we just use some arrays and loops to implement it.

Pseudo Code of Next Fit:

```
int NextFit (int Num, int Item[], int C)
{
        int bin[Num];    //an array to record the contain of each bin, Num items need at most Num bins
        Initial each bin's contain to zero at the beginning;
        for(each bin[i]){
            for(each Item[j]){
                if (Item[j] can be fit in bin[i])
                    bin[i]=bin[i]+Item[j]; //pack Item[j] in bin[i] and process next item
                else (Item[j] cannot be fit in bin[j])
                 break; //go to the next bin to see if it can fit Item[j]
            }
        }
        bin_num = the number of bins whose contains are not zero;
        return bin_num;
}
```

Algorithm 2: First Fit (on-line)

Definition: To scan the bins in order and place the new item in the first bin that is large enough to hold it. Thus, a new bin is created only

when the results of the previous placements have left no other alternative.

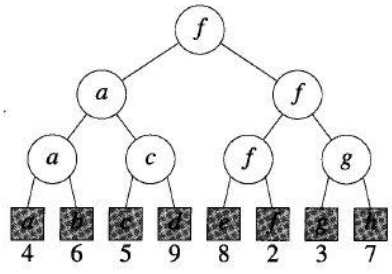
Data structure: to implement the First Fit algorithm in O(NlogN) time complexity, we use a new data structure –Max Winner Tree. The introduction of the Winner Tree is shown as follows:

Winner Tree:

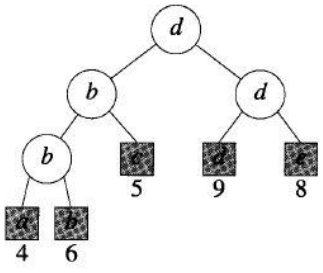
Declaration:

- A winner tree for n players is a complete binary tree with n external nodes and n-1 internal nodes. Each internal node records the winner of each match. To determine the winner, we assume that each player has a value. In a max(min) winner tree, the player who has th larger(smaller) value wins.
- An array t[1] to t[n-1] to represent the internal nodes;
- An array e[1] to e[n] to represent the players.

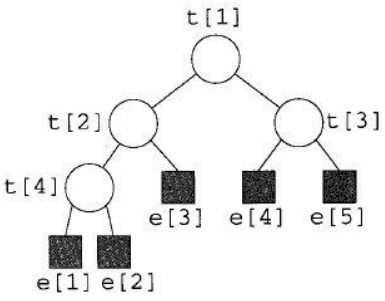
Here are two examples of the winner tree:



a) Min winner tree



b) Max winner tree



Operation:

- Creation(n): to create a winner tree with at most n players;
- Initialize(player[]): to insert the players in the external nodes and play the the whole game.
- Replay: when the value of one player has changed, the game has to been replayed.
- Winner(a,b): to decide the winner of player a and player b;

Advantages:

When the value of one player has changed, it is easy and fast to replay the game in O(logN) time. More detailed analysis of the time complexity is shown in Chapter 4.

Pseudo Code of First Fit:

```
Int FirsttFit (int Num, int Item[], int C)
{
   CreateAWinnerTree(Num);
   Initialize(bin[]=C); //the bins are the players and the available space is the value of each player
   for(Item[i]){
        start from the leftchild of the root T[Q];
            do {
            if (Item[i] can be fit in bin[T[Q]])
                    go to the leftchild of T[Q] and go on checking;
            else (Item[i] cannot be fit in bin[T[Q]]){
```

```
                    go to the rightchild of T[Q] and go on checking;
            } until(T[Q] has no child);
        //after the while loop above, we will finally find the first bin that can fit Item[i], then we pack the Item[j] in the bin
        //bin[T[Q]] is the first fit bin found
                bin[T[Q]]=bin[T[Q]] – Item[i];    //packing the item in the bin
                Replay the game and go on packing the next Item;
     }
     bin_num = the number of bins whose contains are not zero;
    return bin_num;
    }
```

Algorithm 3: Best Fit (on-line)

Definition: To scan the bins in order and place the new item in the tightest bin that is large enough to hold it. Thus, a new bin is created only when the results of the previous placements have left no other alternative.

Data Structure: to implement the algorithm in O(NlogN) time complexity, we use a data structure we have learned before. That is Splay Tree. Since we have learned this data structure before, here we don't introduce the Splay Tree explicitly.

Pseudo Code of Best Fit:

```
int BeststFit(int Num, int Item[],int C)
{
    int bin_num = 0; //record the bin used
```

```
    SplayTree    Tree; //the bins are stored in a splay tree
for(each Item[i]){
    if(Tree==NULL){//no bins in the tree
          Create a bin in the Tree and pack Item[i] in it;
          bin_num++;
          continue; //go on packing the next bin
     }
     find the bin in the tree whose space is closest to Item[i];
     if(Item[i] cannot fit in the bin found){
          Create a bin in the Tree and pack Item[i] in it;
          bin_num++;
          continue;    //go on packing the next Item[i]
     }
     else if (Item[i] exactly fits the bin found)
          Delete the bin from the Tree ; //since the bin is full
     else if (the bin found is a little larger the Item[i]){
          pack the Item[i] in the bin found and Update the Tree;
          continue;    //go to the next item;
     }
}
return    bin_num;
}
```

Algorithm 4: First Fit Decreasing (off-line)

Definition: we scan all the weight of the item and sort them in decreasing order, and then we employ the First Fit algorithm with the sorted items.

Data Structure: to implement this algorithm, we simply sort the items with QuickSort algorithm, and then use the First Fit algorithm.

Pseudo Code of First Fit Decreasing:

```
int FirstFitDecreasing(int Num, int Item[],int C)
{
    QuickSort(Item[]);//sort the items in decreasing order
    bin_num=FirstFit(Num, Item[], C);//employ First Fit algorithm
    return bin_num;
}
```

Chapter 3: Testing Results

In this chapter, we use several test cases to test the running time and bin number of the four different algorithms. What's more, we also test the bin contain level of each algorithm, the bin contain level refers to how full the bin is. Here, the specific testing input data is not included in this report but will be contained in a file called "Test".

- **Testing results for time complexity**

  Total Number: the total number of items to be packed;

  Input: the weight of each item;

Case 1:

When the input is in random order, we test the running time of the four algorithms in different total number.

The testing result is shown in Figure 1 as follows:

Figure 1 **(unit: second)**

TOTAL NUMBER	NEXTFIT	FIRSTFIT	BESTFIT	FIRSTFITDECREASING
100	0.000005	0.000045	0.000048	0.000052
500	0.000019	0.000273	0.000266	0.000304
1000	0.000034	0.000594	0.000547	0.000656
2000	0.000066	0.001314	0.001062	0.001438
3000	0.000103	0.002066	0.001643	0.002297
4000	0.000131	0.002812	0.002188	0.003060
5000	0.000164	0.003670	0.002735	0.004140
6000	0.000206	0.004422	0.003295	0.004898
7000	0.000240	0.005176	0.003852	0.005718
8000	0.000274	0.006000	0.004376	0.006504
9000	0.000295	0.006901	0.004928	0.007604
10000	0.000328	0.007970	0.005470	0.008590
11000	0.000378	0.008678	0.005900	0.009900
12000	0.000394	0.009602	0.006590	0.010542
13000	0.000446	0.010289	0.007184	0.011513
14000	0.000459	0.011225	0.007704	0.012324

Case 2:

When the input is in decreasing order, we test the running time of the four algorithms in different total number.

The testing result is shown in Figure 2 as follows:

Figure 2 **(unit: second)**

TOTAL NUMBER	NEXTFIT	FIRSTFIT	BESTFIT	FIRSTFITDECREASING
1000	0.000028	0.000594	0.002765	0.000656
2000	0.000053	0.001282	0.009218	0.001406
3000	0.000075	0.002066	0.022192	0.002300
4000	0.000106	0.002752	0.038748	0.003000
5000	0.000125	0.003675	0.065855	0.004065
6000	0.000150	0.004428	0.094880	0.004892
7000	0.000197	0.005282	0.125662	0.005831
8000	0.000212	0.005880	0.176000	0.006496
9000	0.000239	0.006892	0.226919	0.007604
10000	0.000266	0.00797	0.29109	0.00859

Case 3:

When the input is in increasing order, we test the running time of the four algorithms in different total number.

The testing result is shown in Figure 3 as follows:

Figure 3 **(unit: second)**

TOTALNUMBER	NEXTFIT	FIRSTFIT	BESTFIT	FIRSTFITDECREASING
1000	0.000030	0.000563	0.000656	0.000641
2000	0.000053	0.001220	0.001624	0.001406
3000	0.000080	0.001970	0.003003	0.002300
4000	0.000100	0.002624	0.005252	0.003000
5000	0.000133	0.003515	0.007345	0.004140
6000	0.000150	0.004235	0.010169	0.004988
7000	0.000186	0.005063	0.013535	0.005718
8000	0.000200	0.005752	0.016000	0.006496
9000	0.000239	0.006622	0.021676	0.007459
10000	0.000265	0.007500	0.025780	0.008600

Case 4:

When the input is in different order, we test the running time of the Next Fit algorithm in different total number.

The testing result is shown in Figure 4 as follows:

Case 5:

When the input is in different order, we test the running time of the First Fit algorithm in different total number.

The testing result is shown in Figure 5 as follows:

Figure 4 **(unit: second)**

TOTAL NUMBER	NEXTFIT INCREASING	NEXTFIT DECREASING	NEXT FIT RANDOM
1000	0.000030	0.000028	0.000034
2000	0.000053	0.000053	0.000066
3000	0.000080	0.000075	0.000103
4000	0.000100	0.000106	0.000131
5000	0.000133	0.000125	0.000164
6000	0.000150	0.000150	0.000206
7000	0.000186	0.000197	0.000240
8000	0.000200	0.000212	0.000274
9000	0.000239	0.000239	0.000295
10000	0.000265	0.000266	0.000328

Figure 5 **(unit: second)**

TOTAL NUMBER	FIRSTFIT INCREASING	FIRSTFIT INCREASING	FIRSTFIT RANDOM
1000	0.000563	0.000594	0.000594
2000	0.001220	0.001282	0.001314
3000	0.001970	0.002066	0.002066
4000	0.002624	0.002752	0.002812
5000	0.003515	0.003675	0.003670
6000	0.004235	0.004428	0.004422
7000	0.005063	0.005282	0.005176
8000	0.005752	0.005880	0.006000
9000	0.006622	0.006892	0.006901
10000	0.007500	0.00797	0.007970

Case 6:

When the input is in different order, we test the running time of the Best Fit algorithm in different total number.

The testing result is shown in Figure 6 as follows:

Figure 6 **(unit: second)**

TOTAL NUMBER	BESTFIT INCREASING	BESTFIT DECREASING	BESTFIT RANDOM
1000	0.000656	0.002765	0.000547
2000	0.001624	0.009218	0.001062
3000	0.003003	0.022192	0.001643
4000	0.005252	0.038748	0.002188
5000	0.007345	0.065855	0.002735
6000	0.010169	0.094880	0.003295
7000	0.013535	0.125662	0.003852
8000	0.016000	0.176000	0.004376
9000	0.021676	0.226919	0.004928
10000	0.025780	0.29109	0.005470

Case 7:

According to our previous analysis, we thought that the size of the bin may affect the running time of the Best Fit algorithm. Here, we provide a test for Best Fit, to compare the running time of Best Fit when bin size is 10e7 and 10, respectively.

The testing result is shown in Figure 7 as follows:

Figure 7 **(unit: second)**

TOTALNUMBER	BESTFIT 1E7	BESTFIT 10
1000	0.000547	0.000203
2000	0.001062	0.000374
3000	0.001643	0.000517
4000	0.002188	0.000752
5000	0.002735	0.00094
6000	0.003295	0.001133
7000	0.003852	0.001324
8000	0.004376	0.001496
9000	0.004928	0.001685
10000	0.00547	0.00188

- **Testing results for Bin Number**

Case 8:

When total number = 100000, we test the number of bins of the four algorithms with input in random order and increasing order respectively. This test case is to compare the performance of the four algorithms in the same level. Also compare them with the ideal number of bins needed.

The testing result is shown is Figure 8 as follows:

Figure 8

ORDER	NEXT FIT	FIRSTFIT	BESTFIT	FIRSTFITDECREASING	IDEAL
Random	66890	50970	50709	50426	50401
Increasing	64598	64598	64598	50519	50444

Case 9:

In different total numbers, we test the bin number of the four algorithms and also compare with the idea number of bin needed. Furthermore, we also test the bin contain level of each algorithm, that is, how full is each bin on average.

The testing result is shown is Figure 9 as follows:

In the following Figure, the data with % refers to the bin contain level, and data beside is the bin number.

Figure 9

TOTAL NUMBER	NEXTFIT		FIRSTFIT		BESTFIT		FIRSTFITDECREASING		IDEAL
10	88.80%	5	88.80%	5	88.80%	5	88.80%	5	5
100	75.65%	62	91.96%	51	91.96%	51	97.71%	48	47
1000	75.68%	684	97.46%	555	98.29%	547	99.62%	543	523
10000	75.31%	6711	98.89%	5211	99.29%	5167	99.78%	5098	5079
100000	75.31%	66815	98.89%	50880	99.29%	50673	99.78%	50428	50316

Chapter 4: Analysis and Comments

Analysis of Case 1, Case 2 and Case 3:

According to the testing result in Figure 1, Figure 2 and Figure3, we get the following Diagram 1, Diagram 2, Diagram 3 and Diagram4, which are showing the running time of the four algorithms with random input, decreasing input and increasing input.

Diagram 1 : Random Input

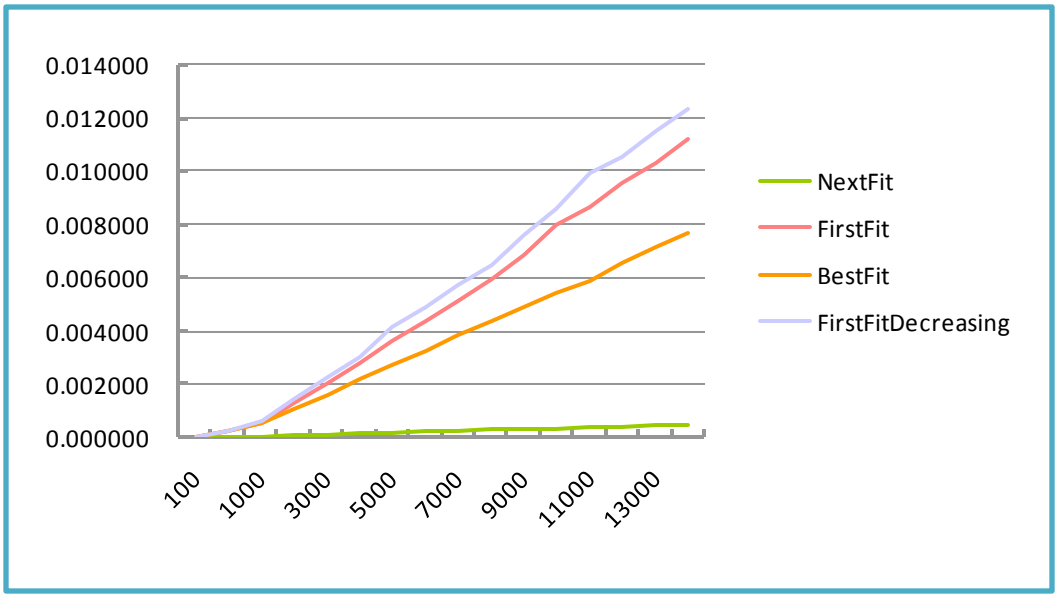


Diagram 2: Decreasing Input

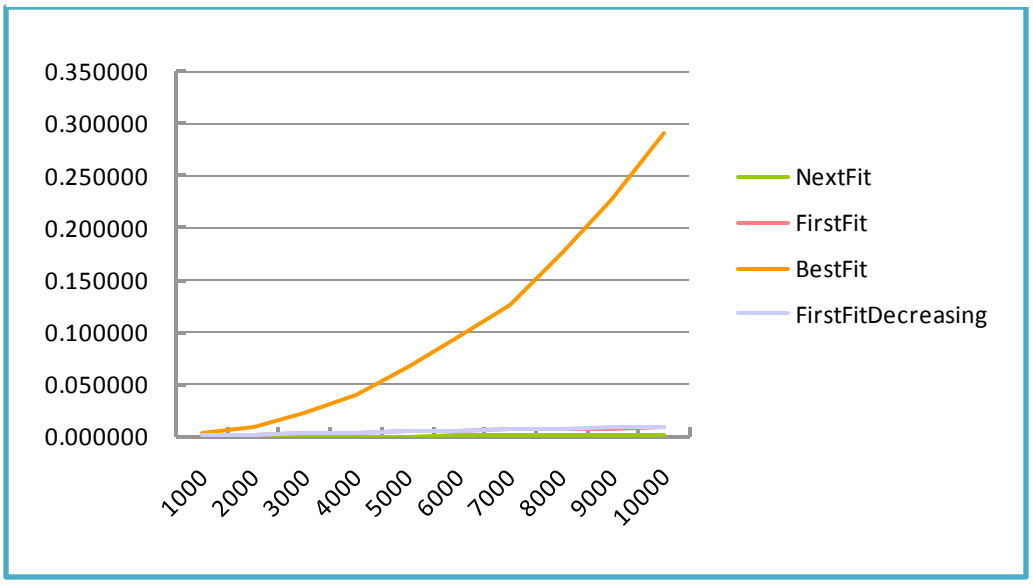


Diagram 3 : Decreasing Input

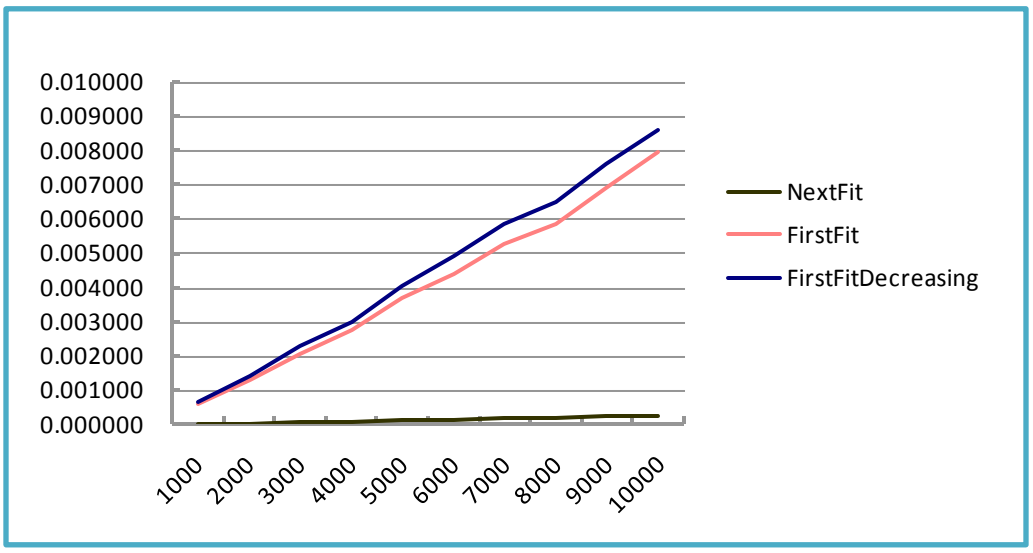
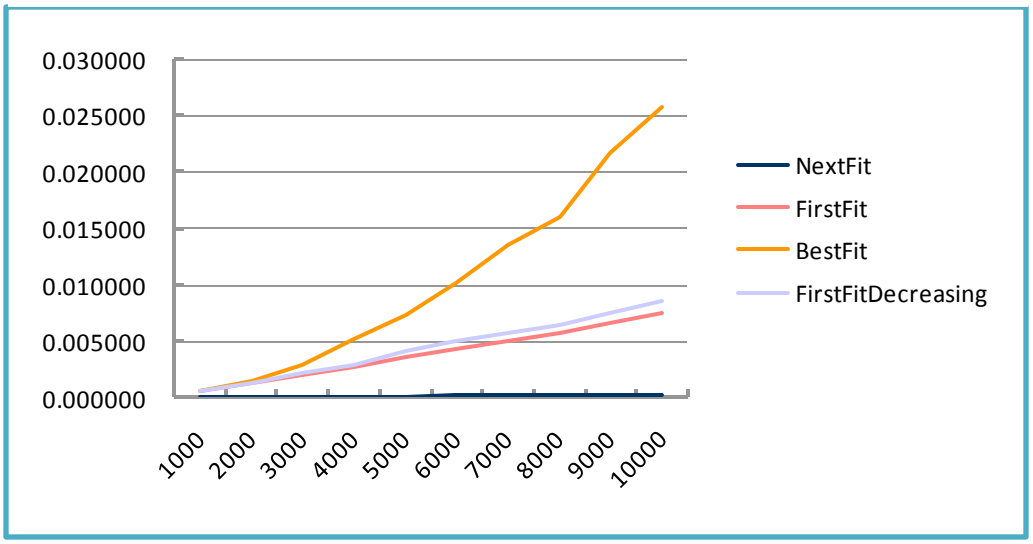


Diagram 4 : Increasing Input

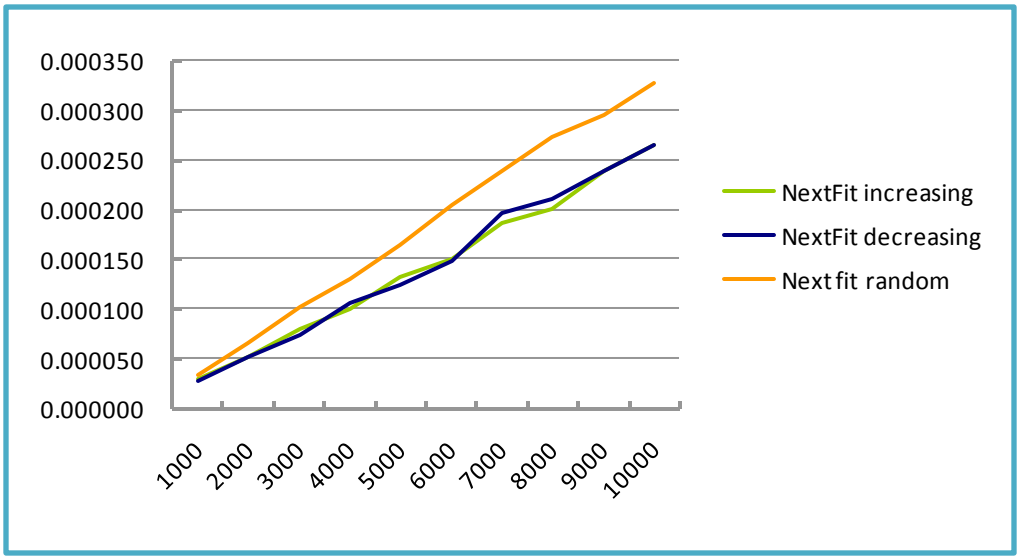


The four diagrams above give us a more intuitively view of the performance of the four algorithms. We can see that , no matter in which input order, the Next Fit always runs the fastest, and much faster than the other three; the First Fit runs nearly the same with the First Fit Decreasing.

Analysis of Case 4, Case 5 and Case 6:

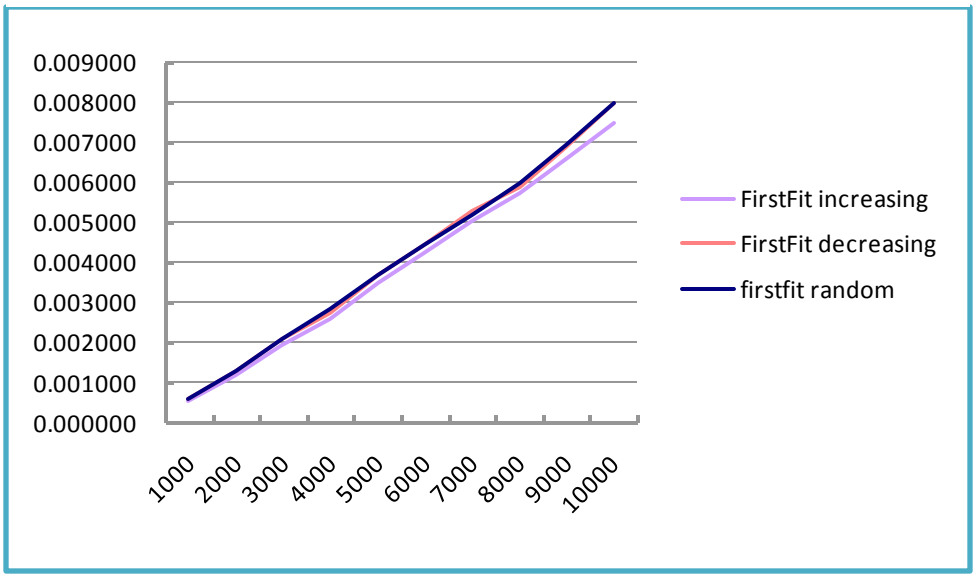
According to the testing result in Figure 4, Figure 5 and Figure 6, we get the following Diagram 5, Diagram 6, Diagram 7, which are showing the running time of Next Fit, First Fit and Best Fit with input in different order respectively.

Diagram 5: Next Fit



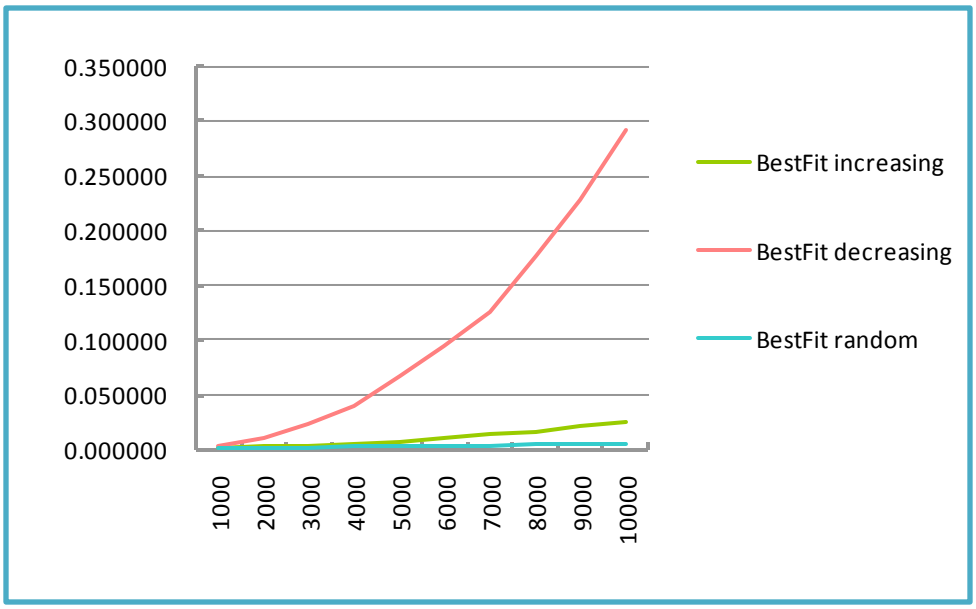
From Diagram 5 above, we see that Next Fit runs almost the same when the input is in increasing order and decreasing order, but runs much more slowly when the input is in random order.

Diagram 6 : First Fit



From the Diagram 6 above, we see that First Fit runs almost the same no matter what order in the input.

Diagram 7 : Best Fit

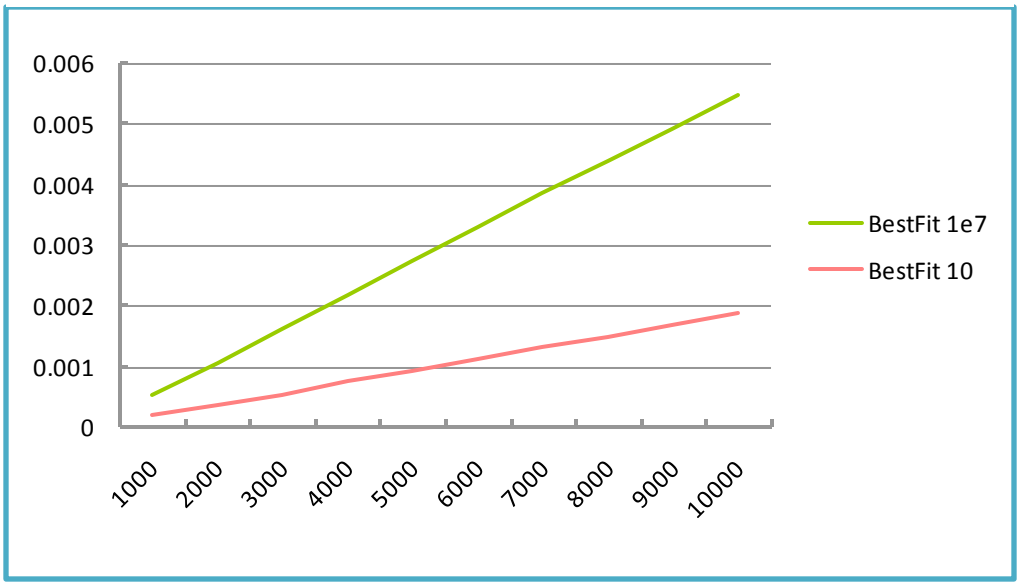


Form Diagram 7 above, we see that Best Fit runs the fastest when the input is in random and runs a little slowly when the input is in increasing order. And when the input is in decreasing order, Best Fit runs much really slowly.

Analysis of Case 7:

According to the testing result in Figure 7, we get the following Diagram 8, which is showing the running time of Best Fit when the bin size is 10e7 and 10 respectively.

Diagram 8 : Best Fit with different bin size

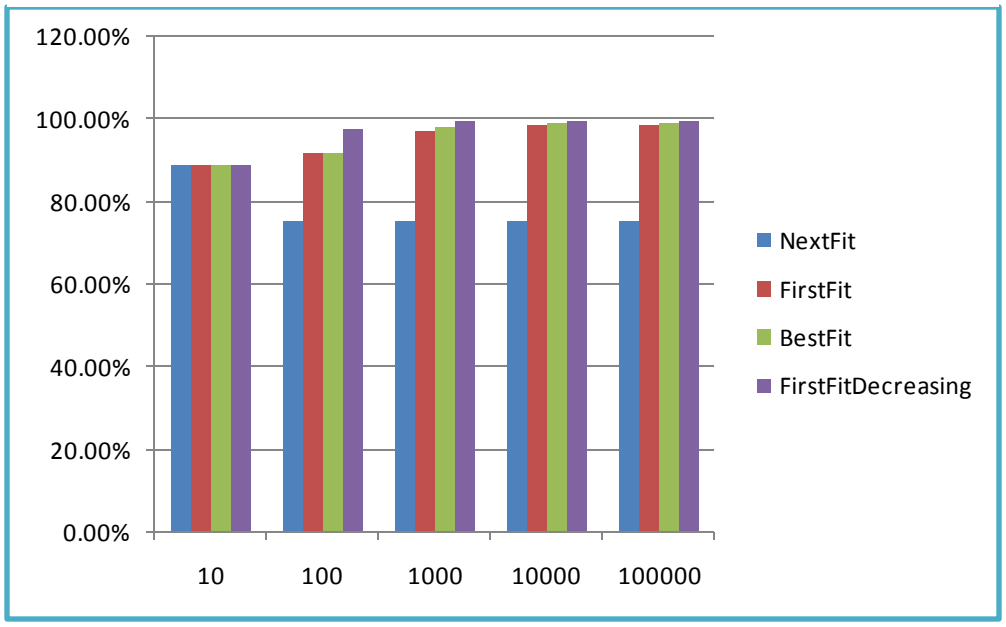


As what we have argued before, how fast the Best Fit runs is related to the bin size. From the Diagram 8 above, we see that , the larger the bin size is ,the more slowly Best Fit runs.

Analysis of Case 8 and Case 9:

According to the testing result in Figure 8 and Figure 9, we get the following Diagram 9, which is showing the bin contain level of the four algorithms with different total number of items.

Diagram 9 : Bin contain level



From the Diagram 9, we see that when the total number is small, the average bin contain level of the four algorithms is almost the same. But when the total number get larger, the difference is more apparent, with that First Fit Decreasing is the best and Next Fit is the worst. The results also reflect that, with the same total weight of all the items, the First Fit Decreasing use the fewest number of bins and the second fewest is Best Fit, and then the First Fit. The Next Fit use the most bins.

Conclusion:

Next Fit

In the next fit program, we use two for loops. And when all the objects are put in the bins and z, which we use to count the number of those objects in the bins, increases to n. In this case, we can never come into the second for loop again, so we can see the results of our test

program, the running time is O(N), N represents the number of the objects. As seen in the figure, the next fit program use more bins, in other words, the bins are not so full compared with other programs. That is because we just create a new bin if the objects are too bigger. We never check there are enough space in the front bins except the last one.

First Fit

We use data type winner tree to build our first fit algorithm. As seen in the source code, winner tree is a full binary tree, we just check from the root to find bins could load our objects.

We ensure it is the first available bin to be used in our code. we just need to change the information on the node on the way from the leaf node which changes values to the root .In this way ,we get a O(NlogN) time complexity. our test data prove it .Of course ,it uses more bins than best fit, and less than next fit.

Best Fit

Best fit implemented with Splay tree runs in O($N \log N$) if the sizes of the elements are all different from each other. When the total number of sizes kinds are far larger than the number of the elements, time complexity of this program is O($N \log N$).Otherwise, if the total number of size kinds are far smaller than the number of the elements, time complexity of this program is near O(N).

So when the total number of size kinds is small, Best fit performs much better than First fit on time.

As it is seen in the figure, best fit use less bins than first fit.

First Fit Decreasing

First fit decreasing is similar to first fit algorithm. But the difference is , the objects must be sorted firstly. So, it uses more time, but because the quick sort does not affect the complexity, it still just needs O(NlogN) time. And because the objects are sorted firstly, it needs the fewest bins.

Appendix: Source Code (in C)

1. WinnerTree.c

```
#include<stdio.h>
#include<stdlib.h>

struct WinnerTree;
typedef struct WinnerTree* WTree;

WTree CreatWTree(int TreeSize);
void  Initialize(WTree WT, int a[],int Size);
void  Play(WTree WT,int k, int lc, int rc);
void  RePlay(WTree WT, int k);
int   Winner(int a[], int b, int c);
int   WinnerT(WTree WT, int m);

struct WinnerTree
```

```
    fdr=FirstFitDecreasing(Total,S,BinSize);
    stop = clock();
    tFD=((double)(stop - start))/CLK_TCK;
    tFD/=FFD_repeat;

/*calculate the average contain of each bin */
    dstN=sum/(double)nr;
    dstF=sum/(double)fr;
    dstB=sum/(double)br;
    dstFD=sum/(double)fdr;

/*print the testing results*/
  printf("NextFit: %d          AVG:%lf      Total time:%lf\n",nr,dstN,tN);
  printf("FirstFit: %d         AVG:%lf      Total time:%lf\n",fr,dstF,tF);
  printf("BestFit: %d          AVG:%lf      Total time:%lf\n",br,dstB,tB);
  printf("FirstFitDecreasing: %d       AVG:%lf   Total time:%lf\n",fdr,dstFD,tFD);

return 1;
}
```

References:

[1]Mark Allen Weiss, “Data Structures and Algorithm Analysis in C,” POSTS&TELECOM PRESS, 2005.

[2] http://en.wikipedia.org/wiki/Winner_tree

[3] http://en.wikipedia.org/wiki/Splay_tree

Author Lists:

Programmer: 黄芯芯 吴蓉 冯焕

Report Writer: 黄芯芯 吴蓉

Tester：冯焕 邵维

PPT maker：邵维

Declaration:

We hereby declare that all the work done in this project titled "Bin Packing Heuristics" is of our independent effort as a group.

Signatures:

黄芯芯

吴蓉

冯焕

邵维