EE3980 Algorithms

HW4 Trading Stock, II

104061212 馮立俞

2018/4/1

Introduction

In this assignment, we're required to improve the brute-force approach for maximum sum array problem in HW03 to reach $O(n^2)$ time complexity. Then, we'll discuss whether it's possible to device an algorithm having lower complexity than $O(n \log n)$.

Approach

Recap: Brute-force Approach

```
1. Algorithm MaxSubArrayBF(A, n, low, high) // Find low and high to
                              maximize ΣA[i ], low≦i ≦high.
2. {
3.      max: = 0;low: = 1;high: = n;
4.      for j: = 1 to n do { // Try all possible ranges: A[j : k ].
5.          for k: = j to n do {
                    sum  = price[high] – price[low];  // n^2 complexity version

6.                  sum: = 0;                       //
7.                  for i: = j to k do {            //
8.                          sum: = sum + A[i];   // n^3 complexity version,
                    }                               // could choose either one

9.                  if (sum > max) then {
                      // Record the maximum value and range.
10.                     max = sum;
11.                     low = j;
12.                     high = k;
13.                 }
14.         }
15.     }
    return max;
  }
```

Since we've known the prices of stock in the given time, the price change over a certain period can be obtained by subtracting the high price with low one. Doing so would save us a loop; thus improve the overall complexity from $O(n^3)$ to $O(n^2)$. Space complexity would remain the same nonetheless, i.e. $O(n)$.

Linear-Complexity Approach

```
1.  Algorithm LinearApproach(Prices[], N) {
2.      max_sum: = 0,
3.      sum: = 0;start: = 0,
4.      end: = 0,
5.      temp_start: = 0;
6.      for i from 0 to N - 1: {
7.          sum += a[i];
8.          if (sum < 0) {
9.              sum: = 0;temp_start: = i + 1;
10.         } //restart record
11.         if (sum > max_sum) {
12.             max_sum = sum;
13.             start = temp_start;
14.             end = i;
15.         } //update record
16.     }
17.     if (start >= end)
18.         do something // array all negative, error handling
19.     return max_sum, start, end;
20. }
```

What is this?

In this implementation, only one single loop is used. Obviously the time complexity is $O(n)$. Plus, no other large memory space is required, so the space complexity is $O(n)$.

?

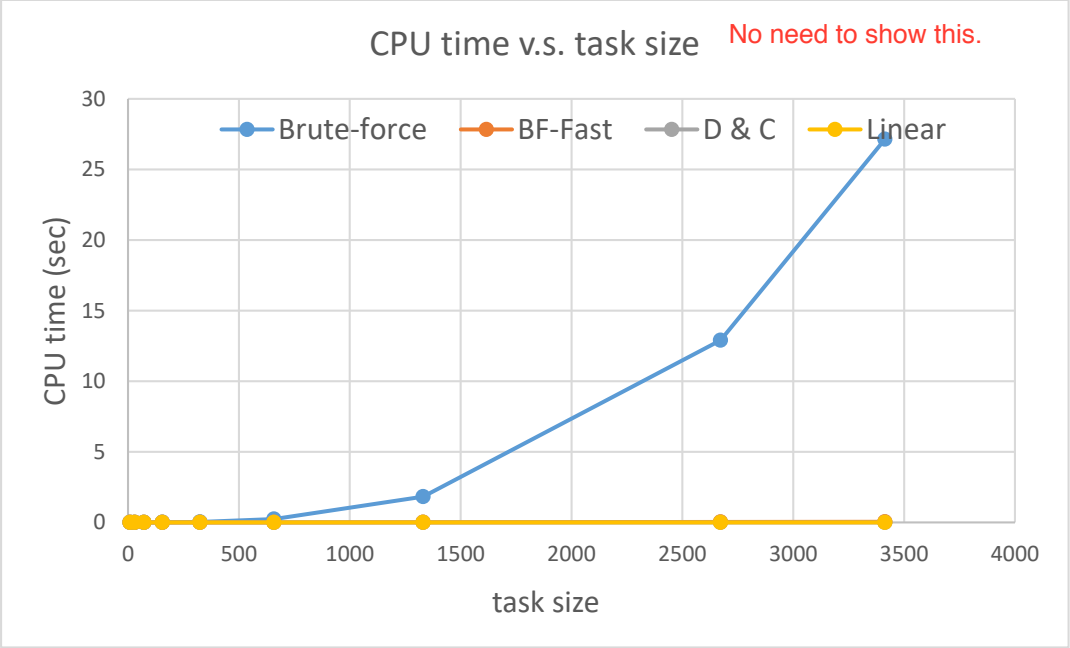
In the algorithm, we keep adding new term to current sum, restart our record if current sum is less than zero, and update `max_sum` and `start & end` if current sum is greater than `max_sum`. Simple as that, yet it works (at least intuitively and empirically for this homework)!

Results and analysis

Table 1. Algorithm CPU time (in seconds) vs. task size

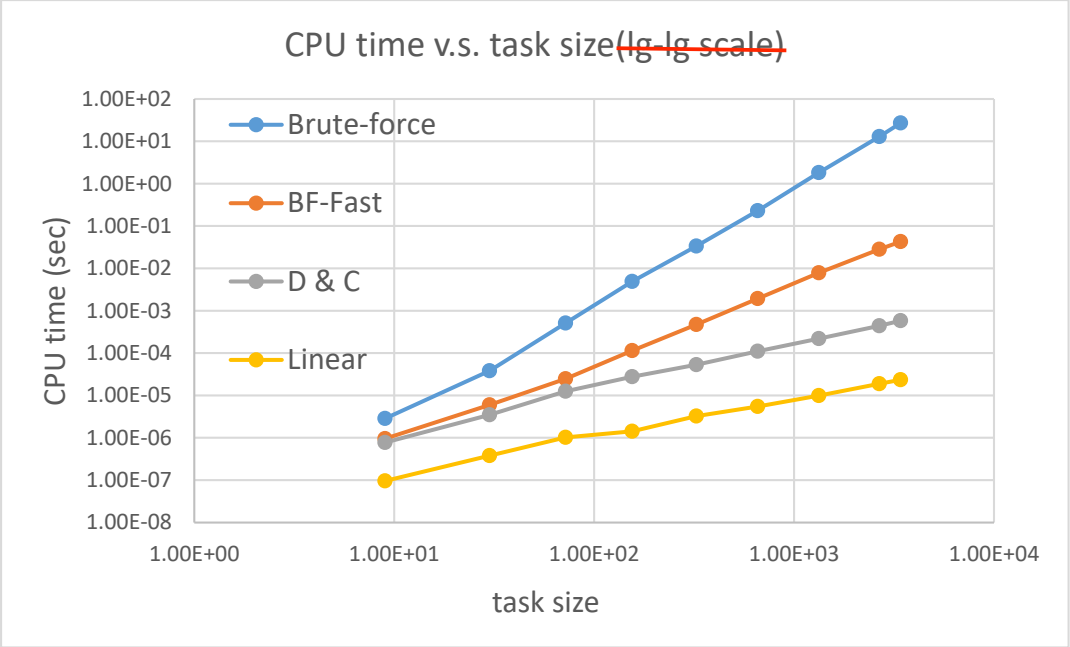
task size	9	30	72	155	325	658	1331	2672	3414
Brute-force	2.86E-06	3.81E-05	0.000509	0.0049	0.0336	0.23	1.82	12.9	27.1378
BF-Fast	9.54E-07	5.96E-06	2.48E-05	1.14E-04	4.73E-04	1.93E-03	7.84E-03	2.83E-02	4.27E-02
D & C	7.72E-07	3.48E-06	1.25E-05	2.76E-05	5.33E-05	1.10E-04	2.19E-04	4.41E-04	5.79E-04
Linear	9.61E-08	3.78E-07	1.02E-06	1.42E-06	3.26E-06	5.46E-06	9.93E-06	1.88E-05	2.35E-05

We can plot above table as follows

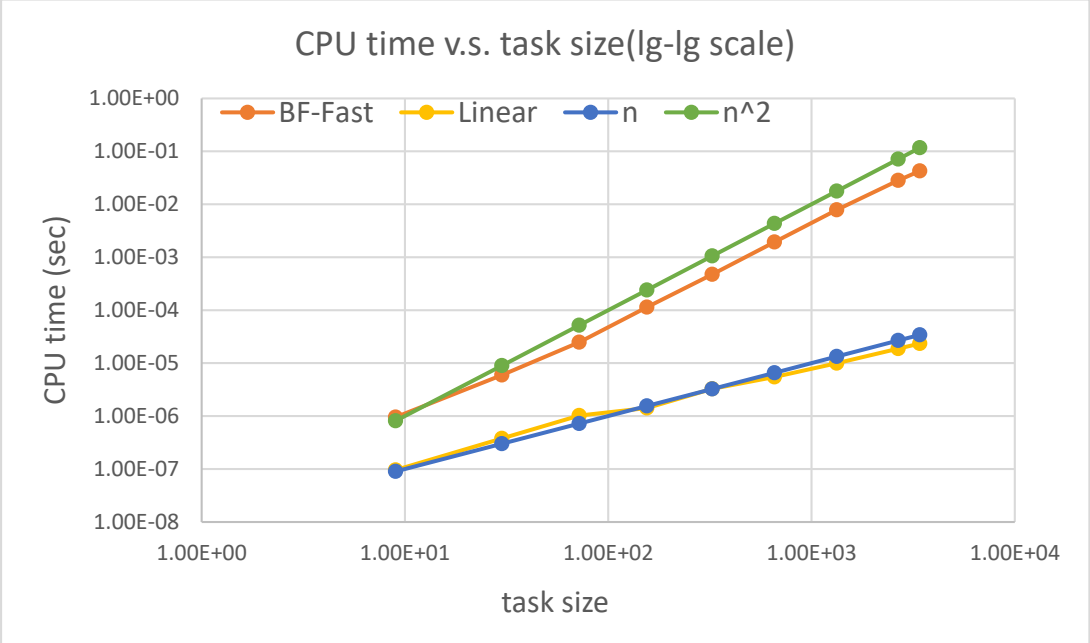


Well, the complexity of the four algorithms are basically not on the same league.

It's pretty hard to plot them on linear scale without some curves being suppressed.



We expect their complexity to be $O(n^3)$, $O(n^2)$, $O(n\log n)$, $O(n)$ respectively. And the curves above are quite fit.



We can further examine the complexity by plotting $O(n^2)$ and $O(n)$ curves with them, and they fit well too.

Observations and Conclusion

"Brevity is the soul of wit", and the same philosophy applies for algorithms too.

With some ingenuity, a great amount of time could be saved.

```
 1 /***************************************
 2   EE3980 Algorithms HW04 Trading Stock, II
 3   Li-Yu Feng 104061212
 4   Date:2018/4/1
 5 ***************************************/
 6
 7 #include <stdio.h>
 8 #include <stdlib.h>
 9 #include <sys/time.h>
10
11 typedef struct sSTKprice {      //store date, price & price change
12     int year,month,day;
13     double price,change;
14 } STKprice;
15
16 typedef struct retval{  //store low, high index and the maximum sum between
17     int low, high;      //for convenient return
18     double sum;
19 } RETval;
20
21 double GetTime(void);
22 void MaxSubArrayBF(STKprice *A, int n);                 //Brute-force method
23 RETval MaxSubArray(STKprice *A, int begin, int end);    //D & C method
24 RETval MaxSubArrayXB(STKprice *A, int begin, int mid, int end);
25                                                        //cross boundary
26 RETval MaxSubArrayFast(STKprice *A, int n);         //linear complexity
27 void PrintResult(STKprice *A, RETval result, double time,int flag);  //as its n
   ame
28
29 double GetTime(void)
30 {
31     struct timeval tv;
32     gettimeofday(&tv,NULL);
33     return tv.tv_sec+1e-6*tv.tv_usec;
34 }
35
36 void MaxSubArrayBF(STKprice *A, int n){
37     double max = 0, sum;
38     double t;               //time
39     int low = 0, high = n-1;
40     int i,j,k;
41
42     t = GetTime();
43     for (j = 0; j < n; j++){        //try all n(n-1)/2 possible situations
44         for (k = j; k < n; k++){
45             sum = A[k].price - A[j].price;  // n^2 complexity version
46             //sum = 0;
47             //for(i = j; i <= k; i++){
48             //  sum += A[i].change;
49             //}                            //n^3 version
```

```
50              if(sum > max){
51                  max = sum;
52                  low = j + 1 ;               //Note: for alignment in printing
53                  high = k;                   //process
54              }
55          }
56      }
57      t = GetTime() - t;
58
59      //print result
60      printf("Brute-force approach:\n");
61      printf("  CPU time %.3g s\n", t);
62      if(low != 0)printf("  Buy: %d/%d/%d at $%g\n",A[low-1].year,
63                      A[low-1].month,A[low-1].day,A[low-1].price );
64      else printf("  Buy: %d/%d/%d at $%g\n",A[0].year,
65                      A[0].month,A[0].day,A[0].price );
66      printf("  Sell: %d/%d/%d at $%g\n",A[high].year,A[high].month,
67                                          A[high].day,A[high].price );
68      printf("  Earning: $%g per share.\n",max);
69  }
70
71
72  RETval MaxSubArray(STKprice *A, int begin, int end){
73
74      int mid;
75      RETval Ans,lret,rret,xret;      //store return values from divided aprts
76      double lsum,rsum,xsum;
77
78      if(begin == end){
79          Ans.low = begin;
80          Ans.high = end;
81          Ans.sum = A[begin].change;
82          return Ans;
83      }
84      mid = (begin + end) / 2;
85      lret = MaxSubArray(A,begin,mid);
86      rret = MaxSubArray(A,mid+1,end);
87      xret = MaxSubArrayXB(A,begin,mid,end);
88
89      lsum = lret.sum;
90      rsum = rret.sum;
91      xsum = xret.sum;
92
93      if(lsum >= rsum && lsum >= xsum){ //left side returns maximum
94          Ans.low = lret.low;
95          Ans.high = lret.high;
96          Ans.sum = lsum;
97      }
98      else if (rsum >= xsum){         //right side
99          Ans.low = rret.low;
```

```
100         Ans.high = rret.high;
101         Ans.sum = rsum;
102     }
103     else{                           //cross boundary
104         Ans.low = xret.low;
105         Ans.high = xret.high;
106         Ans.sum = xsum;
107     }
108     return Ans;
109 }
110
111 RETval MaxSubArrayXB(STKprice *A, int begin, int mid, int end){
112     double lsum = 0,rsum = 0;
113     int low = mid;
114     int high = mid + 1;
115     double sum = 0;
116     int i;
117     RETval Ans;
118
119     for(i = mid; i >= begin; i--){      //find left side max sum
120         sum = sum + A[i].change;
121         if(sum > lsum){
122             lsum = sum;
123             low = i;
124         }
125     }
126     sum = 0;
127     for(i = mid + 1; i <= end; i++){    //find at right side
128         sum = sum + A[i].change;
129         if(sum > rsum){
130             rsum = sum;
131             high = i;
132         }
133     }
134
135     Ans.low = low;
136     Ans.high = high;
137     Ans.sum = lsum + rsum;
138     return Ans;
139 }
140
141 void PrintResult(STKprice *A, RETval result, double time, int flag){
142     int low = result.low;
143     int high = result.high;
144     double sum = result.sum;
145
146     if(flag == 1)   printf("Divide and Conquer approach:\n");
147     else printf("Fast approach:\n");
148     printf("  CPU time %.3g s\n", time);
149     if(low != 0)printf("  Buy: %d/%d/%d at $%g\n",A[low-1].year,    //to avoid
```

```
150                     A[low-1].month,A[low-1].day,A[low-1].price );   //segfault
151     else printf("  Buy: %d/%d/%d at $%g\n",A[0].year,
152                     A[0].month,A[0].day,A[0].price );
153     printf("  Sell: %d/%d/%d at $%g\n",A[high].year,A[high].month,
154                                         A[high].day,A[high].price );
155     printf("  Earning: $%g per share.\n",sum);
156
157 }
158
159 RETval MaxSubArrayFast(STKprice *A, int n){
160     double sum = 0;
161     int temp_start = 0;
162     int i;
163     RETval Ans;
164
165     Ans.sum = 0;
166     Ans.low = 0;
167     Ans.high = 0;
168
169     for (i = 0; i < n; i++){
170         sum +=A[i].change;
171
172         if (sum < 0){
173             sum = 0;
174             temp_start = i+1;
175         }                       //restart record
176
177         if (sum > Ans.sum){
178             Ans.sum = sum;
179             Ans.low = temp_start;
180             Ans.high = i;       //update record
181         }
182     }
183     return Ans;
184 }
185
186
187
188 int main()
189 {
190     int Ndays;
191     int i;
192     double t;
193     STKprice *Prices;
194     RETval result;
195
196     scanf("%d",&Ndays);
197     Prices = malloc(Ndays * sizeof(STKprice));
198
```

```
199     for ( i = 0; i < Ndays; i++){
200         scanf(" %d %d %d %lf", &Prices[i].year, &Prices[i].month,
201                                 &Prices[i].day, &Prices[i].price );
202     }
203
204     Prices[0].change = 0;       //calculate the price changes
205     for ( i = 1; i < Ndays; i++)
206         Prices[i].change = Prices[i].price - Prices[i-1].price;
207
208
209     printf("N = %d\n",Ndays );
210     MaxSubArrayBF(Prices,Ndays);
211
212     t = GetTime();
213     for ( i = 0; i < 1000; i++)
214     {
215         result = MaxSubArray(Prices,0,Ndays-1);
216     }
217     t = GetTime() - t;
218     PrintResult(Prices,result,t/1000, 1);
219
220     t = GetTime();
221     for ( i = 0; i < 1000; i++)
222     {
223         result = MaxSubArrayFast(Prices,Ndays);
224     }
225     t = GetTime() - t;
226     PrintResult(Prices,result,t/1000, 2);
227
228
229     return 0;
230 }
231
232
233
234
235
```

Score: 90

———-

[Pseudo code] on page 3 should be corrected.

- Your also should discuss arrays needed by those two algorithms are different.

[Correctness] of your algorithms should be provided.

- Either rigorous proof or compare the outputs to the old algorithms.