數學校訂必修專題探討 — 繞圈賽

213-11 林孟寬

摘要：

在這次的校訂必修課程中，我們探討如何分配每個人跑的圈數使全部完成秒數最小。從一開始以人工計算觀察可化簡之步驟；因計算繁瑣效率差，想到改以寫程式窮舉列舉所有可能步驟。從結果觀察到窮舉法難以快速計算較大輸入值，課程結束後想到利用暑假所學的貪婪法，使程式更有效率，因而有了以下收穫：

- 利用程式解決問題的能力。
- 思考並使用不同方法解決問題、提升效率。
- 結合先前所學寫程式與演算法，並實際應用於解決生活可見的問題。

指導老師：王香評老師

上課日期：2022 年 10-11 月

1. 動機

學校每年都會舉辦繞圈賽，就是用一個下午的時間，師生們可以一直繞著操場跑並計算圈數，達到指定圈數就可以兌換獎品。而數學校訂必修就結合了繞圈賽的概念，主題是計算一群人如何依照各自的速度來分配每個人的圈數，並在最短的時間內達成指定的圈數。

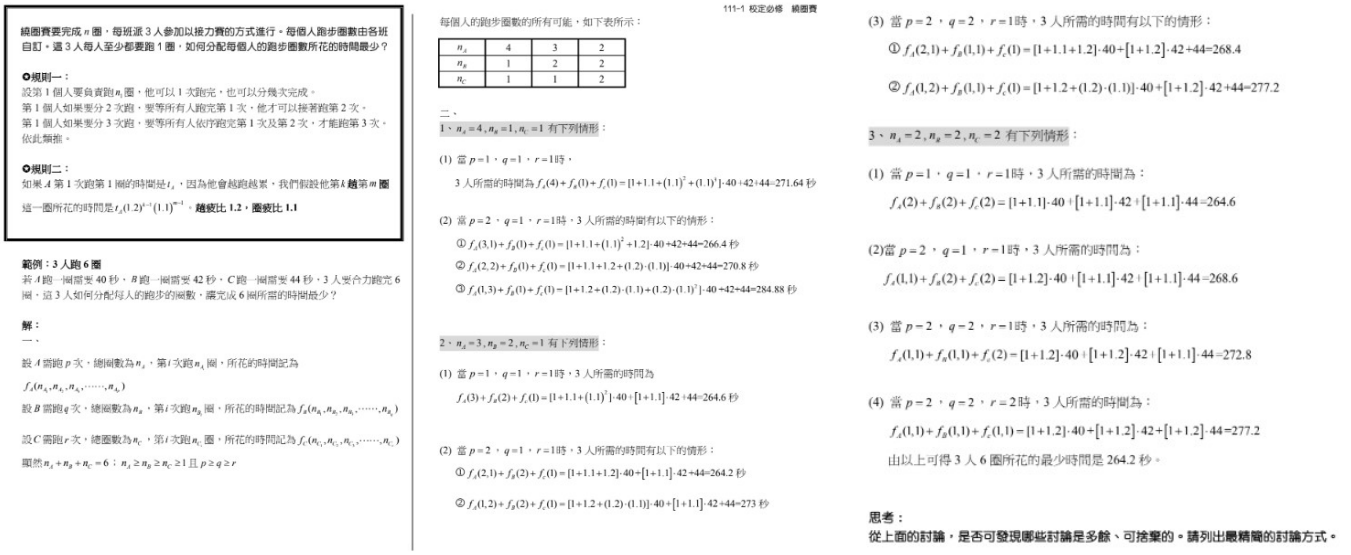
2. 規則

- 繞圈賽要完成n圈，總共派x人以接力賽的方式進行。每個人跑步【整數】圈數自訂，但每人至少都要跑 1 圈，如何分配每個人在每趟 (輪次) 的跑步圈數所花的時間最少？
- 若第 1 個人要負責跑n_1圈，他可以 1 次跑完，也可以分多趟完成。
- 第 1 個人如果要分 2 趟跑，要等所有人跑完第 1 趟，他才可以接著跑第 2 趟。
- 第 1 個人如果要分 3 趟跑，要等所有人都跑完第 1、2 趟才能跑第 3 趟，依此類推。
- 如果第 1 個人第 1 趟跑第 1 圈的時間是t_A。因為他會越跑越累，假設他跑第k趟的第m圈，這一圈所花的時間是$t_A \cdot (1.2)^{k-1} \cdot (1.1)^{m-1}$。

3. 每周進度

3.1. W1-2：研究問題與方法

剛開始兩周先了解題目 (圖 1) 與規則，在簡單的測試範例，先以紙筆列出每一趟可能的人-圈數，觀察是否有一看就知道、秒數明顯過大的假設。接著把剩下的假設一個個算出總秒數，求出最小值。因為使用列舉與人工計算，算個兩、三組就很累了，而且非常沒效率，讓我想用程式來處理，節省計算的時間。



繞圈賽要完成 n 圈，每班派 3 人參加以接力賽的方式進行。每個人跑步圈數由各班自訂。這 3 人每人至少都要跑 1 圈，如何分配每個人的跑步圈數所花的時間最少？

◎規則一：
設第 1 個人要負責跑 n_1 圈，他可以 1 次跑完，也可以分幾次完成。
第 1 個人如果要分 2 次跑，要等所有人跑完第 1 次，他才可以接著跑第 2 次。
第 1 個人如果要分 3 次跑，要等所有人依序跑完第 1 次及第 2 次，才能跑第 3 次。
依此類推。

◎規則二：
如果 A 第 1 次跑第 1 圈的時間是 t_A，因為他會越跑越累，我們假設他第 k **趟**第 m **圈**這一圈所花的時間是 $t_A(1.2)^{k-1}(1.1)^{m-1}$，**趟被比 1.2，圈被比 1.1**

範例：3 人跑 6 圈
若 A 跑一圈需要 40 秒，B 跑一圈需要 42 秒，C 跑一圈需要 44 秒，3 人要合力跑完 6 圈，這 3 人如何分配每人的跑步的圈數，讓完成 6 圈所需的時間最少？

解：
一、
設 A 需跑 p 次，總圈數為 n_A，第 i 次跑 n_{A_i} 圈，所花的時間記為
$f_A(n_{A_1}, n_{A_2}, \cdots, n_{A_p})$
設 B 需跑 q 次，總圈數為 n_B，第 i 次跑 n_{B_i} 圈，所花的時間記為 $f_B(n_{B_1}, n_{B_2}, \cdots\cdots, n_{B_q})$
設 C 需跑 r 次，總圈數為 n_C，第 i 次跑 n_{C_i} 圈，所花的時間記為 $f_C(n_{C_1}, n_{C_2}, \cdots\cdots, n_{C_r})$
顯然 $n_A + n_B + n_C = 6$；$n_A \geq n_B \geq n_C \geq 1$ 且 $p \geq q \geq r$

111-1 校定必修　繞圈賽

每個人的跑步圈數的所有可能，如下表所示：

n_A	4	3	2
n_B	1	2	2
n_C	1	1	2

二、
1、$n_A=4, n_B=1, n_C=1$ 有下列情形：

(1) 當 $p=1$，$q=1$，$r=1$時，
3 人所需的時間為 $f_A(4)+f_B(1)+f_C(1)=[1+1.1+(1.1)^2+(1.1)^3]\cdot 40+42+44=271.64$ 秒

(2) 當 $p=2$，$q=1$，$r=1$時，3 人所需的時間有以下的情形：
① $f_A(3,1)+f_B(1)+f_C(1)=[1+1.1+(1.1)^2]\cdot 40+42+44=266.4$ 秒
② $f_A(2,2)+f_B(1)+f_C(1)=[1+1.1+1.2+(1.2)\cdot(1.1)]\cdot 40+42+44=270.8$ 秒
③ $f_A(1,3)+f_B(1)+f_C(1)=[1+1.2+(1.2)\cdot(1.1)+(1.2)\cdot(1.1)^2]\cdot 40+42+44=284.88$ 秒

2、$n_A=3, n_B=2, n_C=1$ 有下列情形：

(1) 當 $p=1$，$q=1$，$r=1$時，3 人所需的時間為
$f_A(3)+f_B(2)+f_C(1)=[1+1.1+(1.1)^2]\cdot 40+[1+1.1]\cdot 42+44=264.6$ 秒

(2) 當 $p=2$，$q=1$，$r=1$時，3 人所需的時間有以下的情形：
① $f_A(2,1)+f_B(2)+f_C(1)=[1+1.1+1.2]\cdot 40+[1+1.1]\cdot 42+44=264.2$ 秒
② $f_A(1,2)+f_B(2)+f_C(1)=[1+1.2+(1.2)\cdot(1.1)]\cdot 40+[1+1.1]\cdot 42+44=273$ 秒

(3) 當 $p=2$，$q=2$，$r=1$時，3 人所需的時間有以下的情形：
① $f_A(2,1)+f_B(1,1)+f_C(1)=[1+1.1+1.2]\cdot 40+[1+1.2]\cdot 42+44=268.4$
② $f_A(1,2)+f_B(1,1)+f_C(1)=[1+1.2+(1.2)\cdot(1.1)]\cdot 40+[1+1.2]\cdot 42+44=277.2$

3、$n_A=2, n_B=2, n_C=2$ 有下列情形：

(1) 當 $p=1$，$q=1$，$r=1$時，3 人所需的時間為：
$f_A(2)+f_B(2)+f_C(2)=[1+1.1]\cdot 40+[1+1.1]\cdot 42+[1+1.1]\cdot 44=264.6$

(2)當 $p=2$，$q=1$，$r=1$時，3 人所需的時間為：
$f_A(1,1)+f_B(2)+f_C(2)=[1+1.2]\cdot 40+[1+1.1]\cdot 42+[1+1.1]\cdot 44=268.6$

(3) 當 $p=2$，$q=2$，$r=1$時，3 人所需的時間為：
$f_A(1,1)+f_B(1,1)+f_C(2)=[1+1.2]\cdot 40+[1+1.2]\cdot 42+[1+1.1]\cdot 44=272.8$

(4) 當 $p=2$，$q=2$，$r=2$時，3 人所需的時間為：
$f_A(1,1)+f_B(1,1)+f_C(1,1)=[1+1.2]\cdot 40+[1+1.2]\cdot 42+[1+1.2]\cdot 44=277.2$
由以上可得 3 人 6 圈所花的最少時間是 264.2 秒。

思考：
從上面的討論，是否可發現哪些討論是多餘、可捨棄的。請列出最精簡的討論方式。

圖 1：第一周上課講義

3.2. W3-5：以程式處理問題

測試範例：

- 第一題：若 A 跑一圈需要 40 秒，B 跑一圈需要 44 秒，C 跑一圈需要 60 秒，3 人要合力跑完 10 圈，這 3 人如何分配每人的跑步的圈數，讓完成 10 圈所需的時間最少？
- 第二題：若 A 跑一圈需要 40 秒，B 跑一圈需要 44 秒，C 跑一圈需要 48 秒，D 跑一圈需要 52 秒，4 人合力跑完 15 圈，這 4 人如何分配每人跑步的圈數，讓完成 15 圈所需的時間最少？

先寫出計算秒數的程式，輸入每趟要跑幾圈，算出總秒數，節省計算。

```cpp
#include <bits/stdc++.h>
using namespace std;

int main()
{
  ios::sync_with_stdio(0);
  cin.tie(0); cout.tie(0);

  int n; //全部趟數

  int p = 3; //人數
  int time[p] = {40, 44, 60}; //每人秒數

  while (cin >> n)
  {
    int s[100] = {0};
    for (int i = 0; i < n; i++) cin >> s[i];
    //每人圈數，依次為第一人第一趟、第二人第一趟、…、第一人第二趟、第二人第二趟，依此類推

    double ans = 0; //總秒數

    for (int j = 0; j < n; j++)
    {
      for (int i = 0; i < s[j]; i++)
        ans = ans + time[j%p] * pow(1.1, i) * pow(1.2, j/3);
    }

    cout << ans << endl;
  }

  return 0;
}
```

圖 2：計算總秒數的程式 (程式碼連結)

接著用窮舉法來思考程式。

- 想法：總共要跑n圈，用**列出所有加法的方式**，依序代表第幾人第幾趟要跑幾圈。
- 舉例：假設要跑 4 圈，列出所有加法 4、3+1、2+2、2+1+1、1+1+1+1，數字排列的順序和輸入秒數的規則一樣，依序為第一人第一趟的圈數、第二人第一趟的圈數、...、第一人第二趟的圈數、第二人第二趟的圈數，依此類推。
- 列出加法：從n開始，利用遞迴以1為單位分割，直到n＜1。以下程式參考加法分割[1]。

```cpp
#include <bits/stdc++.h>
using namespace std;

void print(vector<int>& v, int level)
{
  for(int i = 0; i <= level; i++)
    cout << v[i] << " ";
  cout << endl;
}

void part(int n, vector<int>& v, int level)
{
  int first;

  if(n < 1) return;

  v[level] = n;
  print(v, level);

  if(level == 0)
    first = 1; //第一次從1開始分割
  else
    first = v[level-1]; //從上次處理完的地方繼續

  for(int i = first; i <= n/2; i++)
  {
    v[level] = i;
    part(n-i, v, level+1); //繼續處理還沒分割過的部分
  }
}

int main()
{
  int num;
  cin >> num;

  vector<int> v(num);

  part(num, v, 0);

  return 0;
}
```

圖 3：列出所有加法程式 (程式碼連結)

最後合併兩個程式。

```cpp
#include <bits/stdc++.h>
using namespace std;

double t(int n, int s[])
{
  int p = 3; //人數
  int time[p] = {40, 44, 60}; //每人秒數
  double ans = 0; //總秒數

  for (int j = 0; j < n; j++)
    for (int i = 0; i < s[j]; i++)
      ans = ans + time[j%p] * pow(1.1, i) * pow(1.2, j/3);

  return(ans);
}

void print(vector<int>& v, int level)
{
  int tmp[10005];

  for(int i = 0; i <= level; i++) tmp[i] = v[i];

  sort(tmp, tmp + level+1);
  do //列出所有加法(依字典排序)
  {
    if(level + 1 >= 3) //每個人都要跑
    {
      cout << level+1 << ","; //io redirect輸出成.csv
      for(int i = 0; i <= level; i++)
        cout << tmp[i] << " ";

      cout << ",";
      cout << t(level+1, tmp);

      cout << endl;
    }
  } while (next_permutation(tmp, tmp + level+1));
}

void part(int n, vector<int>& v, int level)
{
  int first;

  if(n < 1) return;

  v[level] = n;
  print(v, level);

  if(level == 0)
    first = 1; //第一次從1開始分割
  else
    first = v[level-1]; //從上次處理完的地方繼續

  for(int i = first; i <= n/2; i++)
  {
    v[level] = i;
    part(n-i, v, level+1); //繼續處理還沒分割過的部分
  }
}

int main()
{
  ios::sync_with_stdio(0);
  cin.tie(0); cout.tie(0);

  int num;
  vector<int> v(num);
  cin >> num;

  part(num, v, 0);

  return 0;
}
```

圖 4：合併後執行的程式 (程式碼連結)

如果選擇預設的螢幕輸出 (stdio) 不方便觀察可能的解法步驟(如圖 5)，所以在 codeblock 中輸出重新導向 (IO redirection)，輸出.csv 檔 (如圖 6)，並用 Excel 開啟，方便整理結果。

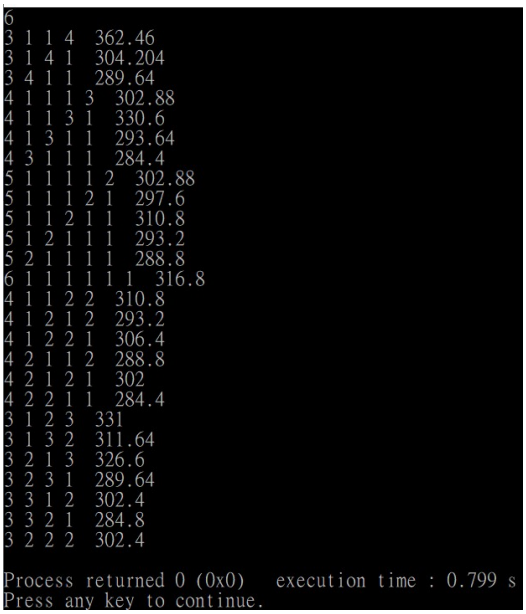


圖 5：螢幕輸出的結果

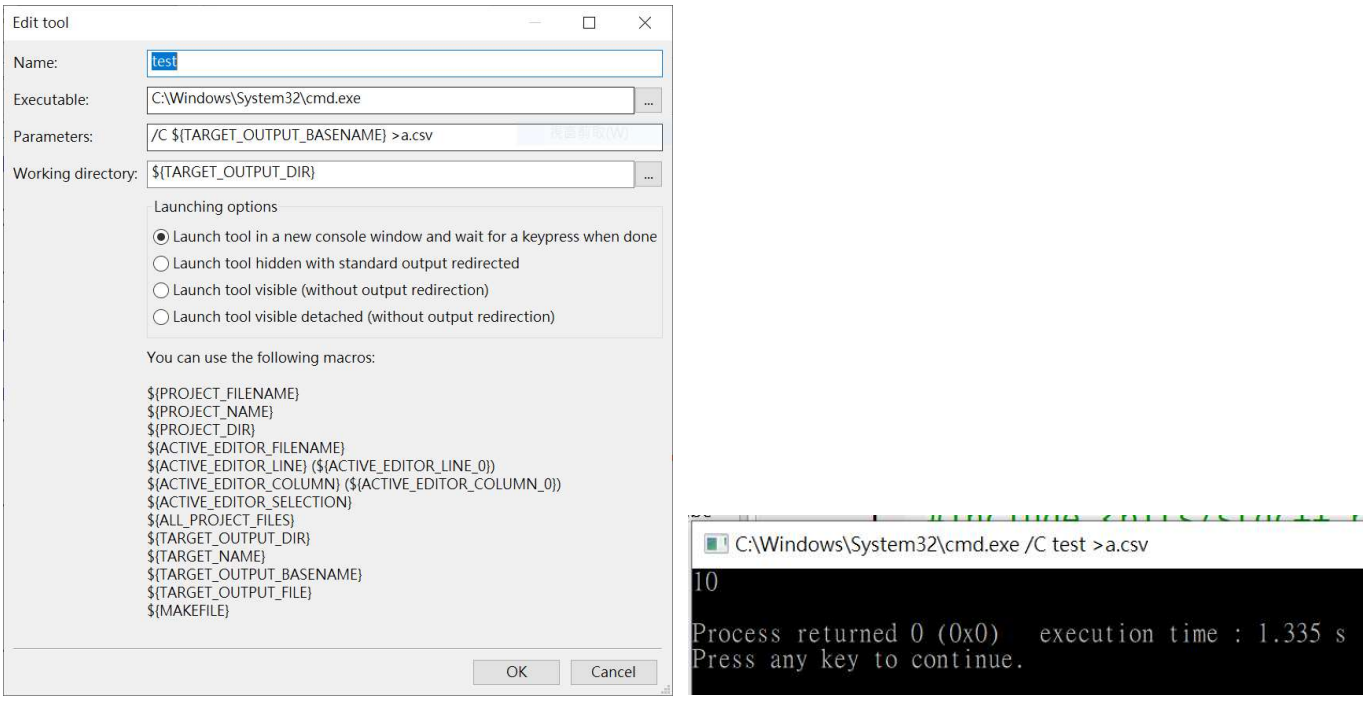


圖 6：在 codeblock 中輸出重新導向參數設定方式 (左圖) 與結果

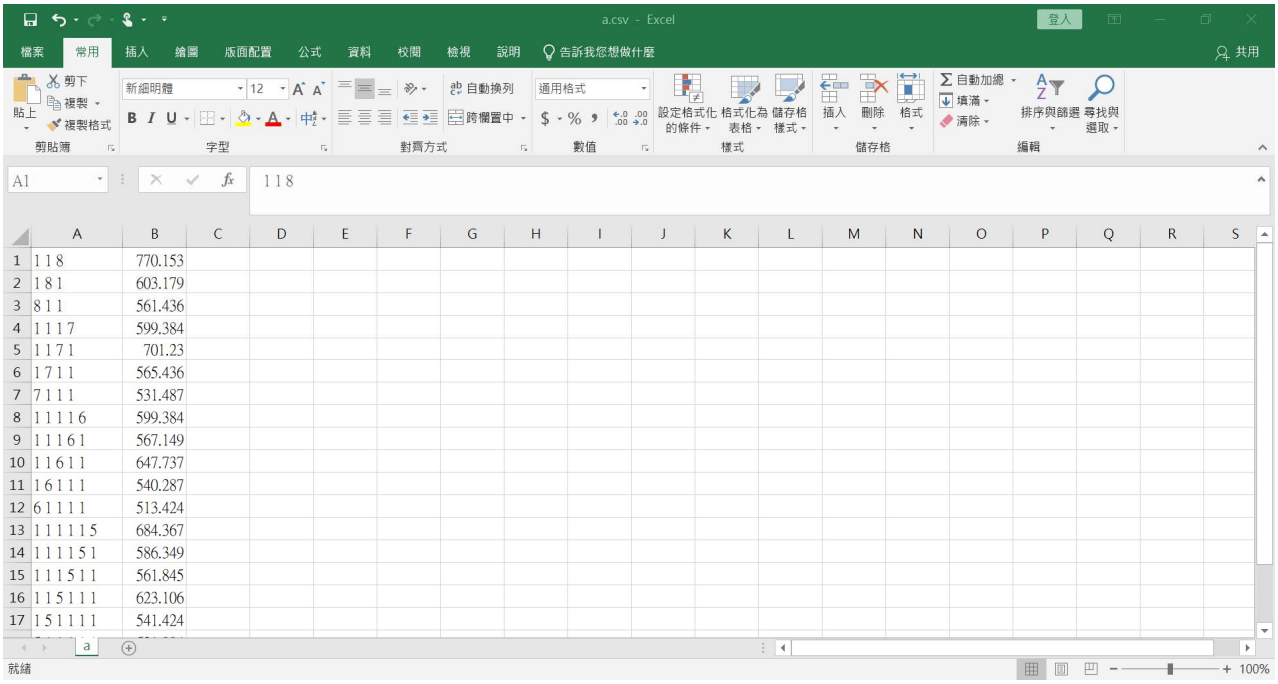


圖 7：以 Excel 開啟結果的.csv 檔

最後再用排序與篩選，就可以快速找出秒數最短的圈數分配方法，並觀察其他分配方式。

測試範例第一題結果：

- 若 A 跑一圈需要 40 秒，B 跑一圈需要 44 秒，C 跑一圈需要 60 秒，3 人要合力跑完 10 圈，這 3 人如何分配每人的跑步的圈數，讓完成 10 圈所需的時間最少？

	A	B
1	42121	491.64
2	33121	491.64
3	43111	492.08
4	4312	492.08
5	32131	496.48
6	32122	496.48
7	42112	496.92
8	33112	496.92
9	4213	496.92
10	3313	496.92

圖 8：第一題輸出結果

- 如圖 8，使用最少秒數的分配方式有兩個最佳解，分別為 4 2 1 2 1 或 3 3 1 2 1，用了 **491.64** 秒。

測試範例第二題結果：

- 若 A 跑一圈需要 40 秒，B 跑一圈需要 44 秒，C 跑一圈需要 48 秒，D 跑一圈需要 52 秒，4 人合力跑完 15 圈，這 4 人如何分配每人跑步的圈數，讓完成 15 圈所需的時間最少？

	A	B
1	4 3 2 2 2 1 1	752.48
2	4 3 2 2 3 1	752.96
3	4 3 3 2 2 1	752.96
4	4 3 2 2 2 2	752.96
5	4 3 2 1 3 1 1	753.36
6	4 3 3 1 2 1 1	753.36
7	4 3 2 1 2 2 1	753.36
8	5 3 2 2 2 1	753.444
9	4 4 2 2 2 1	753.444
10	4 3 3 1 3 1	753.84

圖 9：第二題輸出結果

- 如圖 9，使用最少秒數的分配方式為 4 3 2 2 2 1 1，用了 **752.48** 秒

4. 課後：貪婪法

課程結束後，我對這個題目仍很感興趣，想用不同方法再處理一次問題，上課時可以用窮舉暴搜是因為數字很小才能跑出結果，不過圈數只要大於 20，就需要花很多時間才跑得出結果，我對這樣的結果很不滿意，我這次決定用比窮舉更有效率的方式再做一次。想到暑假為了 APCS 在網路上自學的貪婪法：在每個狀態下選擇最佳解，可以用在求最小值的問題。所以決定用貪婪法寫，並檢驗自己的學習。

想法：

- 先開兩個陣列紀錄每趟的圈數跟秒數，依序代表第一人第一趟、第二人第一趟、…、第 n 人第一趟、第一人第二趟、第二人第二趟、…、第n人第k趟，依此類推。
- 以窮舉時加法分割來思考，每次都只跑一圈時要跑的次數最多，也就是 1+1+1…+1，所以陣列大小只要開跟圈數一樣就好。
- 紀錄秒數的陣列初始為該趟一圈的秒數。例如第一人第一趟第一圈為 a 秒，則第一人第二趟第一圈為$a \times 1.2$秒，第一人第三趟第一圈為$a \times 1.2 \times 1.2$秒，依此類推。
- 貪婪法即在當下選擇最佳解，所以每次都選擇最小的秒數，並把這趟的圈數+1。選擇後也代表此人這一趟多跑一圈，故記錄秒數的陣列中此項要再乘上1.1。
- 因為每人至少要跑一圈，所以先直接把每個人第一趟第一圈的秒數跟圈數加上去。
- 如果要跑第n趟，要等所有人都完成$n-1$趟才能繼續，所以如果在陣列中，該趟的前一項是0，代表前面的人還沒跑過，所以此解不合，要繼續找第二佳的答案。

程式碼與執行成果

```cpp
#include <bits/stdc++.h>
using namespace std;

bool cmp(pair<int, double> a, pair<int, double> b)
{
  return a.second < b.second;
}

int main()
{
  ios::sync_with_stdio(0);
  cin.tie(0); cout.tie(0);

  int p, n; //人數, 圈數
  cin >> p >> n;
  int csize = n; //最多要跑幾趟
  int c[100005] = {0}; //第幾人第幾趟對應跑的圈數
  double a[1005]; //每人單圈秒數
  for(int i = 0; i < p; i++) cin >> a[i];
  double ans = 0; //總秒數
  pair<int, double> s[100005]; //第幾人第幾趟, 對應的單圈秒數

  for(int i = 0; i < n; i++)
  {
    s[i].first = i;
    s[i].second = a[i%p] * pow(1.2, i/p);
  } //先算出每人每趟第一圈的秒數

  for(int i = 0; i < p; i++)
  {
    ans += a[i];
    s[i].second *= 1.1;
    c[i]++;
  } //每人第一趟的第一圈先加上去
  n -= p;

  while(n > 0)
  {
    sort(s, s + csize, cmp); //找出最小秒數

    int tmp = 0; //紀錄這次最佳解的編號
    bool v = 0; //紀錄這個解是否合理
    while(v == 0) //如果不合就繼續找下一個解
    {
      if(s[tmp].first >= p && (c[s[tmp].first-1]) == 0)
        tmp++; //編號+1，找下一個最佳解
      else v = 1;
    }

    ans += s[tmp].second; //總秒數加上去
    c[s[tmp].first]++; //這一趟的圈數+1
    s[tmp].second *= 1.1; //這趟多跑了一圈，秒數乘上去
    n--; //還要處理的圈數-1
  }

  for(int i = 0; i < csize; i++)
    if(c[i]) cout << c[i] << " ";
  cout << ans << endl;

  return 0;
}
```

圖 10：貪婪法程式碼 (程式碼連結)

- 如圖 11 圖，當圈數變大後 (25 圈)，貪婪法的效率變很明顯了。

```
4 25
40 44 48 52

5 4 3 3 3 2 2 1 1 1 1353.49

Process returned 0 (0x0)   execution time : 0.020 s
Press any key to continue.
```

```
1353.49

Process returned 0 (0x0)   execution time : 28.122 s
Press any key to continue.
```

圖 11：貪婪法 (左) 與窮舉 (右) 的執行時間差了超過 1,400 倍

5. 心得與反思

在這堂數學課中，我更感受到程式的方便之處，以程式解決問題減少了許多繁瑣的紙筆計算，更結合了數學邏輯的美妙。雖然一開始用很沒效率的窮舉法，但也因為效率差到我不能接受，才再想起還不夠熟悉的貪婪法並加以應用，除了試著用不同方法思考問題，也驗證了自己的想法、更精熟學習成果，並在改進的過程中獲得滿滿成就感。

6. 參考資料

[1] 整數分割槽、分割槽數：https://www.796t.com/post/N2IzdGs=.html