Algorithm Hw1

Group 3

* Question 1:

1. Bubble sort:

* Stable：因為排序方式為從頭一個個比大小，當遇到大小相同的值時就不會 swap，所以會維持其輸入的先後。
* In-place：演算法中純粹做左右交換，不需要而外的空間去儲存，頂多需要幾個變數暫存，又其空間複雜度為 O(1)。

1. Insertion sort
   * Stable：當新的值再插入時若遇到相同到小的值會將其放在已經排好的值的右邊，所以先後順序不會改變（即先輸入的仍然在左邊）。
   * In-place：在插入時可以維持在原本的陣列中排序，不需要而外的空間，又其空間複雜度為 O(1)。
2. Selection sort
   * Unstable：

＊Counter-example：

Initial：4a, 2, 3, 4b, 1 (Swap 1 & 4a)

First step：1, 2, 3, 4b, 4a (become unstable)

* In-place：僅需在原陣列中做 swap，不需要另外的空間儲存，頂多需要幾個變數暫存，又其空間複雜度為 O(1)。

1. Quick sort
   * Unstable：

＊Counter-example：

Initial：4a, 2, 3, 4b, 1 (pivot is 3, so swap 4a & 1)

First step：1, 2, 3, 4b, 4a (become unstable)

* In-place：可以在原先陣列中 swap，所以可以算是 in-place。

1. Merge sort
   * Stable：通常在 merge 時的規則是 L ≤ R，所以遇到相同大小的值時並不會進行 swap，原本在左邊的值（即先輸入的）仍會維持在相對左邊的位置。
   * Not-in-place：傳統的 merge sort 需要額外的空間來儲存 merge 的結果，所以為 not-in-place，其空間複雜度為o(n)。
2. Heap-sort

* Unstable：

＊Counter-example：

Initial：5, 4a, 4b, 3, 2, 1 (Max heap)

(Start sorting, delete max)

Step 1：4a, 3, 4b, 1, 2 |5

Step 2：4b, 3, 2, 1 |4a, 5

Step 3：3, 1, 2 |4b, 4a, 5

Step 4：2, 1 |3, 4b, 4a, 5 (左右相對順序交換)

In-place：僅需要在原先建立的 tree 中進行 swap 與 heapify，不需要額外的空間，所以為 in-place。

* Question 2:

1. Solution 1:

int main()

{

int i , n;

scanf("%d",&n);

i = 0;

while (i\*i<=n)

i++;

printf("%d\n",i-1);

return 0;

}

從i=0開始做到i\*i>n，最後再回傳i-1。時間複雜度O(√ n)。

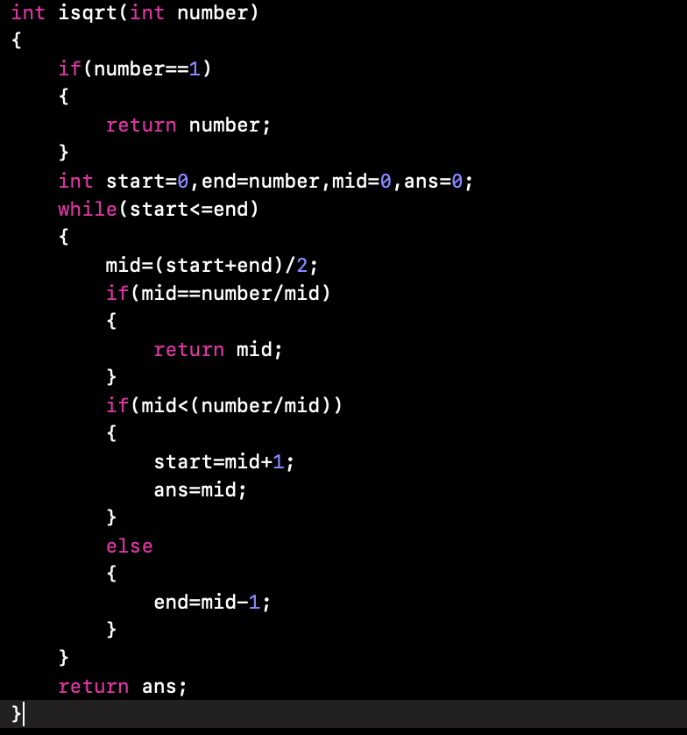
1. Solution 2(better solution):

圖2-1. Code of solution2

* 利用二元搜尋的方法，從下限start=0,end=number開始。
* 當start<=end持續做

計算mid=(start+end)/2;

比較mid\*mid和number

如果mid\*mid等於number return mid;

如果mid\*mid小於number 將start=mid+1 ; ans=mid;

如果mid\*mid大於number 將end=mid-1 ;

* Question 3:
* 長度為n的陣列A中選出n/2個最小的元素，依序排至長度為n/2陣列B中。
* 將剩餘未被選取的元素排序在陣列A的後半部。
* 將陣列B的元素覆蓋陣列A的前半部。

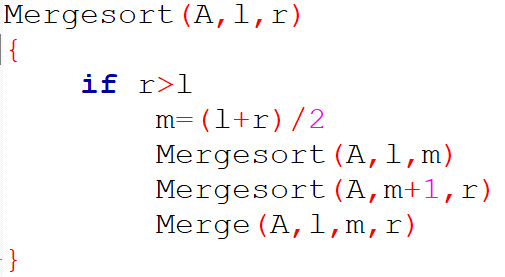
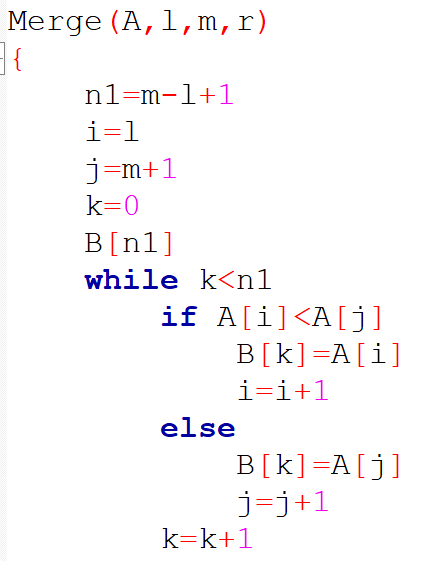
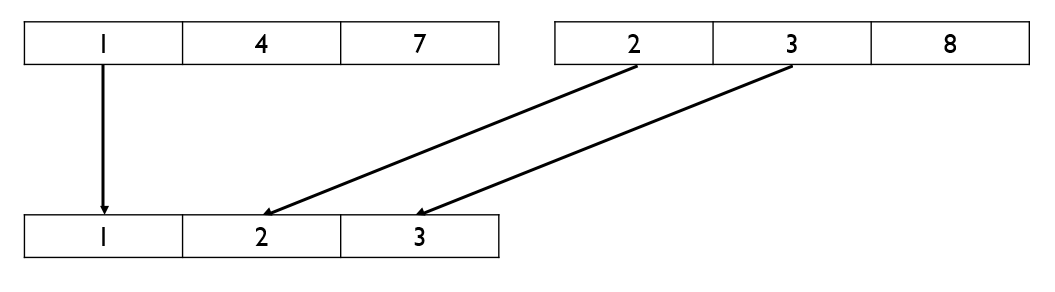
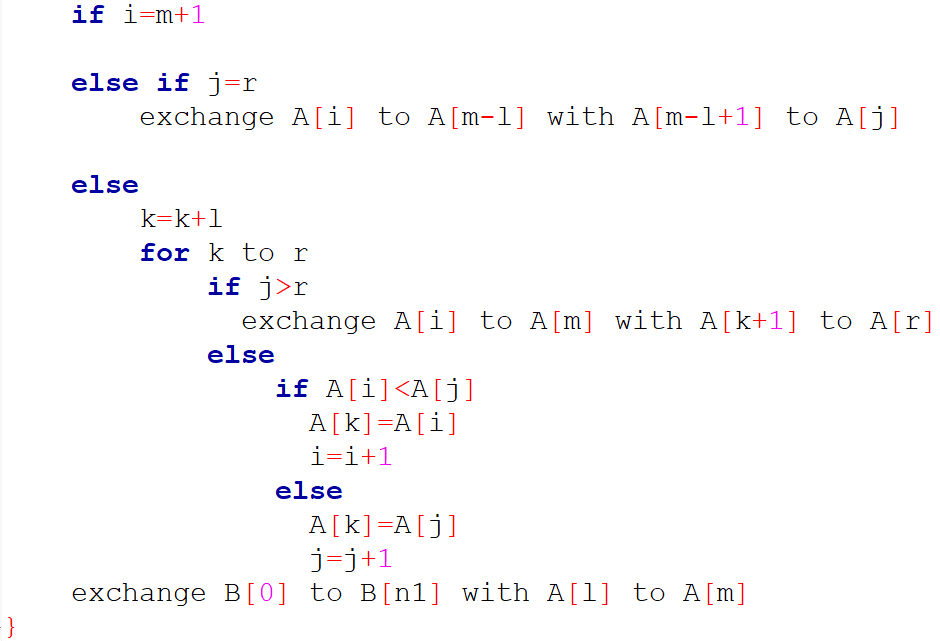
 圖3-1. New merge sort 示意圖

圖3-2. Pseudocode of new mergesort

* Question 4:

a[i],b[j] \*/Two be sorted set/\*

* Compute the union of two sets:

while (I < a.length & j < b.length)

if a[i] < b[j];

c[k]=a[i];

i++;

k++;

else if a[i] > b[j]

c[k]=b[j];

j++;

k++;

else if a[i] == b[j]

c[k]=a[i];

i++;

j++;

k++;

return (c[k]);

* Compute the intersection of two sets

while( i<a.length & j< b.length){

if (a[i]<b[j] ){

i++;}

else if (a[i]>b[j] ){

k++;}

else if (a[i]=b[j] ){

c[k]=a[i];

i++;

k++;}

}

return (c[k]);

* Determine if a given element is in a given set

for(i=0;i<= a.length ;i++){

if(element = a[i]){

return true;

}}

* Question 5:

由於兩個array已經排序過了，假設由小排到大，可以用merge的步驟找到 。

int ans = 0, i = 0, j = 0;

ans = abs(x[0] - y[0]); **//初始 | x[i]− y[j]| 的min值**

while (i < m && j < n) {

if (abs(x[i] - y[j]) < ans) {

ans = abs(x[i] - y[j]);  **// 如果(x[i] - y[j])的值小於ans, 把差值設為新ans值**

}

if (x[i] < y[j]) { **//如果Y那列比較大，X那列的index就往下一個**

i++;

}

else if (x[i] > y[j]) { **// 如果X那列比較大，Y那列的index就往下一個**

j++;

}

else {

ans = 0; **// 如果差值是0，可以直接跳出**

break;

}

}

return ans;

圖5-1. Code of solution 5

* 時間複雜度：考慮worst case，每個元素都要被比較，總共做m+n-1次比較，所以是O(m+n)。
* Question 6:

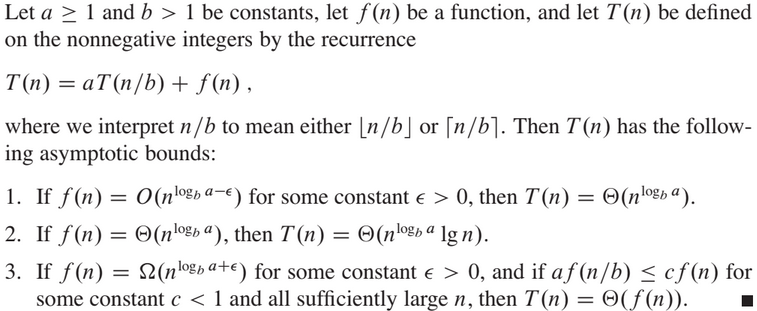
By master theorem

圖6-1. Master theorem

let

then

So

* Question 7:

i = 2

while i <= size

if i == 1 or array[i] >= array[i - 1]

i += 1

else

swap array[i], array[i - 1]

i -= 1

此排序演算法為bubble sort

* + best-case  
    當數列為排好時，即為best-case   
    [1, 2, 3, …, n-1, n]，只需要比n-1次，不須用swap，因此time complexity=O(n)。
  + worst-case  
    當數列為由大排到小時（順序相反），即為worst-case，time complexity=O(n2)  
    [5, 4, 3, 2, 1] -> i的index= 2, 1, 2 ，比3次 -> [4, 5, 3, 2, 1]  
    [4, 5, 3, 2, 1] -> i的index= 3, 2, 1, 2, 3 ，比5次 -> [3, 4, 5, 2, 1]  
    得出：移動第n個數字需要比2n-1次  
    排序n個數字= 3 + 5 + 7 + … + 2n-1  
    全部加起來為n2-2n+1，因此time complexity=O(n2)。
  + average-case  
    將第k個數放到正確位置可能需要比較1, 3, 5, 7, … ,2k-1次，假設機率為uniform distribution  
    則期望值為1\*(1/k) + 3\*(1/K) + … + (2k-1)\*(1/k) = (1 + 2k-1)\*k /(2k) = k  
    排序n個數字 => ，因此time complexity=O(n2)。