Khuyen Le Thi Minh s5128 – ASD 2

## TASK DESCRIPTION:

Let A and B are an incrementally ordered arrays of natural numbers and K be some arbitrary natural number. Find an effective algorithm which determines all possible pairs of indexes (i,j) such that A[i]+B[j]=K. Prove algorithm's
correctness and estimate its complexity.

## ALGORITHM WITH LINEAR SEARCH:

```
Pairing1(int [] A, int[] B, int a, int b, int x)
{
          for i:=0 to a do                                   //iterative loop
                   for j:= 0 to b do                         //iterative loop
                              if(A[i]+B[j] == x) then        // checking condition
                                       return Pair(i,j);     //return result
                              fi;
                   od;
         od;
}
```

## Justification of an algorithm's correctness:

The specification of the algorithm Pairing1 becomes a pair <WP, WK>, where WP = (A and B are an incrementally ordered arrays of natural number, a, b, x are also a natural number, where a = A' length – 1 and b = B's length --- 1) and WK = (A[i]+B[j] = x, for i and j are also natural numbers).

<u>Stop condition of the algorithm</u> – at the beginning j=0 and in each iteration of the for loop variable j is increment by 1, exactly after b+1 iteration, j = b+1 and the condition (for j:=0 to b) is not satisfied in the loop. Thus the inner for loop stops.
At the beginning i=0 and in each iteration of the for loop variable i is increment by 1, exactly after a+1 iteration, i = a+1 and the condition (for i:=0 to a) is not satisfied in the loop. Thus the algorithm stops for any data satisfying the initial condition WP.

<u>Partial correctness of the algorithm</u> – for the initial condition WP = (A and B are an incrementally ordered arrays of natural number, x is also a natural number) and WK = (A[i]+B[j] = x, for I and j are also natural numbers) we get:

```
Pairing1(int [] A, int[] B, int a, int b, int x)          // WP = (A and B are an incrementally ordered arrays
of natural number, x, a, b are natural numbers and a=A's length – 1 and b = B's length --- 1)
{
     for i:=0 to a do                      //loop control ------> i=0, when i < A length, i increase by 1
          for j:= 0 to b do                                                  //loop control ------> j=0, when
j < B length, j increase by 1
               if(A[i]+B[j] == x) then                                 // comparing: A[i]+B[j]=x and i<=a
and j<=b
                    return Pair(i,j);                       //WK=( A[i]+B[j]=x)
               fi;
          od;
     od;
}
```

Hence the formula A[i]+B[j] = x is an invariant of the loop in the algorithm Pairing1, and if it terminates the comparison, it will return Pair(i,j) if the end condition WK=( A[i]+B[j] = x) is true.

Because the Pairing1 algorithm is partially correct in terms of specification <WP,WK> and we have shown the stop condition for this algorithm, it is also absolutely correct with respect to the considered specification.

## Estimation of algorithm's complexity:

Time complexity – assuming that the dominant operation in the Pairing1 is the natural numbers addition and comparison, then T(a+1, b+1) = a*b = O( $n^2$)

Space complexity – the Pairing1 algorithm uses two auxiliary variables i and j independently of the value of the argument a and b, therefore S should be S(a+1, b+1) = O(1).

## ALGORITHM COMBINING BINARY AND LINEAR SEARCH:

```
getBound(int[] A, int a, int k)
{
        int first := 0, last := a, result = 0;                    //variable initialisation
          while (first <= last) do                          //iterative loop
                   int mid:= first+(last---first)/2;
                    if (A[mid] == k) then
                               result = mid;
                             first := mid + 1;
                    else if (A[mid] > k) then
                             last := mid – 1;
                     else then
                             first := mid + 1;
                              result = first;
                    fi;
           od;
          return result;                                    //return result
}

Pairing2(int [] A, int[] B, int a, int b, int x)
{
          int y0 := x – A[0], x0 := x – B[0];                     //variable
initialisation        int yn := getUpperBound(A, a, y0), xn := getUpperBound(B, b, x0);
           for i:=0 to xn do                                //iterative loop for j:= yn to 0 do
                                                //iterative loop
                              if(A[i]+B[j] == x) then           // checking condition return Pair(i,j);
                                                //return result
                             fi;
          od; od;
}
```

## Justification of an algorithm's correctness:

The specification of the algorithm Bound becomes a pair <WP, WK>, where WP = (A is an incrementally ordered arrays of natural number, k and a are also a natural number and a = A's length --- 1) and WK = (result: A[result] >= k>A[result – 1]).

Stop condition of the algorithm – at the beginning first=0 and in each iteration of the while loop variable first is increment by 1, exactly after last+1 iteration, j = last+1 and the condition (for j:=0 to last) is not satisfied in the loop. Thus the inner for loop stops. Thus the algorithm stops for any data satisfying the initial condition WP.

Partial correctness of the algorithm – for the initial condition WP = (A is an incrementally ordered arrays of natural number, a and k are also a natural number, where a = A's length --- 1) and WK = (result: A[result + 1] > A[result] >= k>A[result – 1]) we get:

```
getBound(int[] A, int a, int k)
{
                    if (A[mid] == k) then
                               result = mid;
                             first := mid + 1;
                    else if (A[mid] > k) then
```

```
                    last := mid – 1;
              else then
                    first := mid + 1;
              fi;

      int first := 0, last := a, result = 0;                       //last = a, where a = A's length --- 1
        while (first <= last) do                      //loop control ------> first=0, when first< A's length, first
increase by 1
              int mid:= first+(last---first)/2;              //finding the middle element position of the array
              if (A[mid] == k) then                          //comparing A[mid] with k, if A[mid]==k
                     result = mid;                           //we temporary keep the result value
                    first := mid + 1;                        // we discard the left side of the array
              else if (A[mid] > k) then                      //comparing A[mid] with k, if A[mid]>k
                    last := mid – 1;                         // we discard the right side of the array
               else then                                     //comparing A[mid] with k, if A[mid]==k
                    first := mid + 1;                        // we discard the left side of the array
                            result = first;                            //we temporary keep the
              result value fi;
        od;
        return result;                                      // returning  result or 0 if there is no result

}
```

Hence the formula A[mid] < A[next] and A[mid] == A[next]  are the invariant of the loop in the algorithm getBound, and if it terminates the comparison, it will return 0 if there is no result and result if the end condition WK = (result: A[result] >= k>A[result – 1]) is true.

Because the getBound algorithm is partially correct in terms of specification <WP,WK> and we have shown the stop condition for this algorithm, it is also absolutely correct with respect to the considered specification.

The specification of the algorithm Pairing2 becomes a pair <WP, WK>, where WP = (A and B are an incrementally ordered arrays of natural number, a, b, x are also a natural number, where a = A' length – 1 and b = B's length --- 1) and WK = (A[i]+B[j] = x, for i and j are also natural numbers).

<u>Stop condition of the algorithm</u> – at the beginning j=yn and in each iteration of the for loop variable j is decrement by 1, exactly after yn+1 iteration, j = ---1 and the condition (for j:=yn to 0) is not satisfied in the loop. Thus the inner for loop stops.

At the beginning i=0 and in each iteration of the for loop variable i is increment by 1, exactly after xn+1 iteration, i = xn+1 and the condition (for i:=0 to xn) is not satisfied in the loop. Thus the algorithm stops for any data satisfying the initial condition WP.

<u>Partial correctness of the algorithm</u> – for the initial condition WP = (A and B are an incrementally ordered arrays of natural number, x is also a natural number) and WK = (A[i]+B[j] = x, for I and j are also natural numbers) we get:

```
Pairing2(int [] A, int[] B, int a, int b, int x)
{
        int y0 := x – A[0], x0 := x – B[0];                            //variable initialisation, getting the
highest values     int yn := getUpperBound(A, a, y0), xn := getUpperBound(B, b, x0); //getting bounds
         for i:=0 to xn do                          bound, // loop control ------> i=0, when i <= A's upper
i increase by 1
                for j:= yn to 0 do                  j       // loop control ------> j=B's upper bound, when j
        decrease by 1                                       >=0,
                         if(A[i]+B[j] == x) then            // comparing: A[i]+B[j]=x and i<=xn and j<=yn
                                return Pair(i,j);                 //WK=( A[i]+B[j]=x), we can print
                       directly i and j here, the formula Pair(i,j) is just for algorithm to be more easier
        to read. fi; od; od;
}
```

Hence the formula A[i]+B[j] = x is an invariant of the loop in the algorithm Pairing1, and if it terminates the comparison, it will return Pair(i,j) if the end condition WK=( A[i]+B[j] = x) is true.

Because the Pairing2 algorithm is partially correct in terms of specification <WP,WK> and we have shown the stop condition for this algorithm, it is also absolutely correct with respect to the considered specification.

## Estimation of algorithm's complexity:

Time complexity – assuming that the dominant operation in the getBound is the natural numbers comparison, then the number of operation that it has to make is T(a+1) = O(n)

Assuming that the dominant operation in the Pairing2 is the natural numbers addition and comparison, then T(a+1, b+1) = )(a+1) + O(b+1) = O(2n) = O(n) with n is the bigger number between *a* and *b*.

Space complexity – the get Bound and Pairing2 algorithms use auxiliary variables: first, last, mid, next, g, y0, x0, yn, xn, i and j independently of the value of the argument a and b, therefore S should be S(a+1, b+1) = O(1).