56 3, 7, 9, 9, 11, 17, 29, 85.

Algorithm for Selection Sort: sorting the array in ascending order.

~~for i ← 1 to n-1 to do~~

```
for i ← to n-1 do
    min j=i;
    min x ← A[i]
for j ← i+1 to n do
    if A[j] < min x then
        minj ← j
        minx ← A[j]
    end
end

A[minj] ← A[i]
    A[i] ← min x
end
```

Insertion Sort:

Algorithm Insertion Sort(A)

```
for j = 2 to A Length
    Key = A[j]
while i > 0 and A[i] > Key
    A[i+1] = A[i]
      i = i-1
    end
   A[i+1] = Key
end
```

Algorithm (Merge Sort) sorted

Algorithm merge-sort (A, low, high)
if low < high then
mid ⇐ floor (low high) / 2
merge sort (A, mid+1, high)
combine (A, low, mid, high)
end
combine (A, low, mid, high)
l_1 = mid - low + 1
l_2 = high - mid.
for i = 1 to l_1 do

```
for i = 1 to l1 do
    Left [i] ← A [low + i - 1]
end.

for j = 1 to l2 do
    Right [j] ← A [mid + j]
end.
for K = low to high do
    if left [i] ≤ Right [j] then.
        B [K] ← left [i]
        i = i + 1.
    else
        B [K] ← Right [j]
        j = j + 1
    end
end
```

=

l_2 =

① low ← 1
high ← n
while low < high do
mid = $\frac{low + high}{2}$

Iteration method
B Search

②
if A[mid] = Key then
Return mid
else if A[mid] < Key then
low = mid+1
else
high = mid-1
end
end
Return 0.

	1	2	3	4	5	6	7	8
A =	11	22	33	44	55	66	77	88

Key = 33.

Finding Minimum and Maximum.

Algorithm max-min (A [1.....n], max, min)

```
    max ← min ← A [1]
    for (i ← 2 to n ) do
    {
      if (A[i] > max) then
        max ← A[i]   // Obtaining max value
      if (A[i] < min ) then
        min ← A[i]  // Obtaining min value
    }
}
```

Greedy-Fractional-Knapsack (w(1....n), p(1...n), w)
weight = 0
for i = 1
if weight + w[i] ≤ w then
{
n[i] = 1
weight = weight + w[i]
}
else {
n[i] = w - weight / w[i]
}
weight = w
break
return n
}
En:
w = 60

```
Algorithm JOB_SCHEDULING( J, D, P )
// Description: Schedule the jobs using greedy approach
which maximizes the profit
// Input:   J: Array of N jobs
        D: Array of deadline for each job
        P: Array of profit associated with each job
Sort all jobs in J in decreasing order of profit

                                    If job does not miss
                                    its deadline

S ← Ø // S is set of scheduled jobs, initially it is empty
SP ← 0   // Sum is the profit earned
                              Add job to solution set
for i ← 1 to N do
   if Job J[i] is feasible then
      Schedule the job in latest possible free slot meeting its
deadline.
      S ← S ∪ J[i]
      SP ← SP + P[i]              Add respective profit
   end
end
```

```
Algorithm FLOYD_APSP ( L)
// L is the matrix of size n X n representing original graph
// D is the distance matrix
D ← L
for k ← 1 to n do
    for i ← 1 to n do
        for j ← 1 to n do
            D[i, j]^k ← min ( D[i, j]^(k-1), D[i, k]^(k-1) + D[k, j]^(k-1) )
        end
    end
end
return D
```

Algorithm:

```
sumofsubset(s,k,r)
{
        X[k]=1;
        if (s+W[k]=m) then write(X[1:k]);
        else if (s+W[k]+W[k+1]<=m)
         then sumofsubset(s+W[k], k+1,r- W[k]);

        if ((s+ r-W[k]>=m)and(s+ W[k+1]<=m)) then
        {
                X[k]=0;
                sumofsubset(s, k+1, r- W[k]);

        }
}
```

described below :

```
Algorithm  NAÏVE_STRING_MATCHING(T, P)
// T is the text string of length n
// P is the pattern of length m
for i ← 0 to n – m do
    if P[1... m] == T[i+1...i+m] then
        print "Match Found"
    end
end
```

Complexity analysis