

1. If $z = (z_0, \dots, z_{n-1})$ is a list of n complex numbers, the *Fourier transform* of z is another list $Z = (Z_0, Z_1, \dots, Z_n)$ of n complex numbers, where

$$Z_k = \frac{1}{\sqrt{n}} \sum_{i=0}^{n-1} z_i \omega^{ik} \quad (1)$$

where ω is the *primitive n th root of unity*, the complex number $\cos(2\pi/n) + \mathbf{i} \sin(2\pi/n)$. Note that $\omega^n = 1$, and therefore $\omega^k = \omega^{k \bmod n}$ for any integer k .

The Fourier transform has many extremely useful properties, but they do not matter for this equation. What matters is that computing the Fourier transform by formula (1) would require n^2 complex number multiplications and almost as many complex number additions. The following algorithm, known as *fast Fourier transform* (FFT), is much more efficient, in theory and practice. It requires n to be a power of 2, $n = 2^r$.

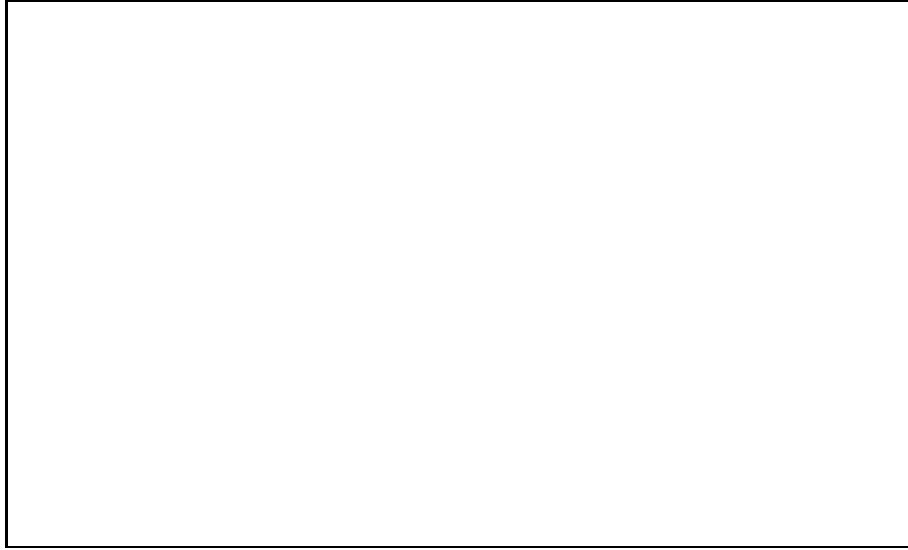
```

Algorithm FFT( $n, z, \omega$ ) returns  $Z$ 
  if  $n = 1$  then return  $z$ ; endif
  // Split even and odd:
   $h \leftarrow n/2$ 
  for  $i$  from 0 to  $h - 1$  do
     $z'_i \leftarrow z_{2i}$ ;
     $z''_i \leftarrow z_{2i+1}$ ;
  endfor;
  // Recurse fro each subsequence:
   $Z' \leftarrow \text{FFT}(h, z', \omega^2)$ 
   $Z'' \leftarrow \text{FFT}(h, z'', \omega^2)$ 
  // Combine th transforms:
   $\sigma \leftarrow 1$ 
  for  $k$  from 0 to  $h - 1$  do
    [g]  $Z''_k \leftarrow \sigma Z''_k$ ;
         $Z_k \leftarrow Z'_k + Z''_k$ ;
         $Z_{k+h} \leftarrow Z'_k - Z''_k$ ;
         $\sigma \leftarrow \sigma\omega$ ;
  endfor;
  return  $Z$ ;

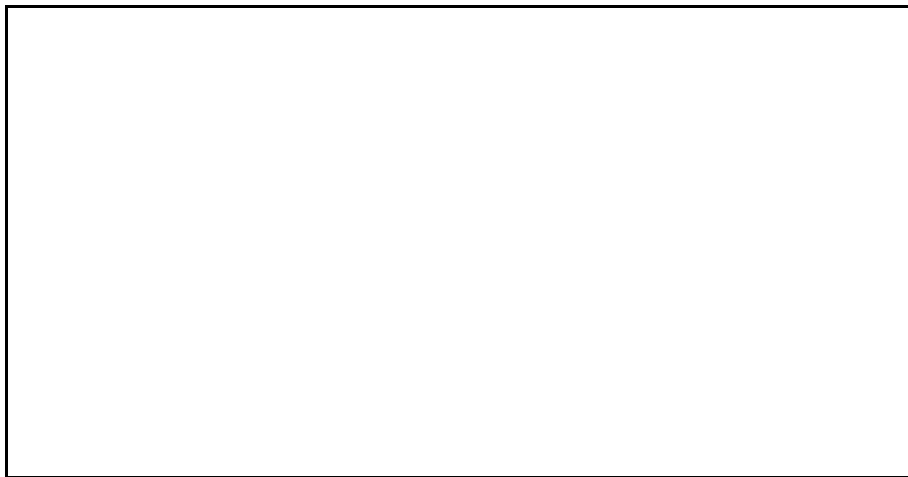
```

Note that, the algorithm executes 2 complex multiplications and 2 complex additions for each execution of the statement labeled *g*. (We can ignore the computation of ω^2 in the arguments to the recursive call, since it can be taken from a precomputed table.). Assuming that n is a power of 2:

- (a) Give recurrence formulas for the total number of multiplications $\mu(n)$ and the total number of additions $\alpha(n)$ performed by *FFT*, including operations inside the recursive calls.



- (b) Find explicit (non-recursive) formulas for $\mu(n)$ and $\alpha(n)$.



2. The following recursive algorithm $Permute(n, x, k, use)$ generates all permutations of the elements of a list $x = (x[0]..x[n - 1])$ of n arbitrary elements, leaving the first k elements fixed.

In particular, if $k = 0$ the procedure generates all permutations; if $k = n$, it generates a single permutation, namely the given list x itself. For each generated permutation, $Permute$ calls the function use , provided by the calling program, with arguments (n, x) . The elements of x are rearranged while call is in progress, but, at the end, they will be restored to their original order.

For example, if $n = 5$, $x = (07, 97, 27, 35, 45)$, and $k = 2$, then $Permute(n, x, k, use)$ will call

```
use(5, (07, 97, 27, 35, 45))
use(5, (07, 97, 27, 45, 35))
use(5, (07, 97, 35, 27, 45))
use(5, (07, 97, 35, 45, 27))
use(5, (07, 97, 45, 35, 27))
use(5, (07, 97, 45, 27, 35))
```

and the list x will again contain $(07, 97, 27, 35, 45)$.

```
Algorithm  $Permute(n, x, k, use)$ 
  if  $k = n$ 
    [ $u$ ]    $use(n, x)$ ;
  else
    for  $i$  from  $k$  to  $n - 1$  do
      [ $f$ ]   swap  $x[k] \leftrightarrow x[i]$ ;
             $Permute(n, x, k + 1, use)$ ;
            swap  $x[k] \leftrightarrow x[i]$ ;
    endfor;
  endif;
```

