235 (longest palindrome) A palindrome is a list that equals its reverse. Write a program to find a longest palindromic segment (sublist of consecutive items) in a given list.

After trying the question, scroll down to the solution.

§ If we had functions with local declarations, there would be an easy solution as follows. (In this solution, whenever I say "segment" I means a pair of indexes.)

Define a binary-valued function *pal* that, given a segment of the list, says whether the segment is a palindrome. It works from the ends to the middle so that we don't need to make separate cases for odd and even length segments.

Now write a recursive function *longestpal* that, given a segment of the list, finds a longest palindrome in the segment. It calls *pal* for the segment. If *pal* says yes, then return the segment. If *pal* says no, then the segment must be at least two items long, so call *longestpal* for the segment minus its first item. Save the returned segment. Then call *longestpal* for the segment minus its last item. Compare the saved segment with the newly returned segment, and return the longer (either if a tie).

Call *longestpal* for the whole list. If n is the length of the list, the time $t\,n$ as a function of n is such that $t\,n + t\,n = t(n+1)$. So time $\le 2^n$.

If we don't have recursive functions with local declarations, then "Save the returned segment." means pushing it onto a stack, and "Compare the saved segment" means popping it off the stack.

If n is the length of the list, there are about n starting points for a segment, and at each starting point there are on average $n/2$ segments, and on average each segment is $n/2$ items long, and to check a segment of length $n/2$ to see if it is a palindrome requires at most $n/4$ comparisons. So we shouldn't need more than $n^3/8$ time units. With a little thought, we can do even better because all segments with the same center can be checked together working outward from the center, and at the first length that is not a palindrome, we know that all longer segments with that same center are also not palindromes. That should bring the time down to $n^2/4$. And we shouldn't need a stack. So let's try again.

Let the given list be L . Let c , p , q , i , and j be natural variables. Variables p and q will be the indexes of the longest palindromic segment $p,..q$ found so far. Variables i and j will be the indexes of the segment $i,..j$ currently under investigation. Variable c will be twice the index of the center of the current segments under investigation. If c is even, then the current segments are even length segments centered at $c/2$ (drawing indexes between items as always). If c is odd, then the current segments are odd length segments whose middle item is $L((c-1)/2)$. Variable c begins at 0 , increases by 1 , and stops increasing when the distance from the center of the current segments to the end of the list is less than or equal to half the length of the longest palindrome so far. We stop when $\#L - c/2 \le (q-p)/2$, or $2\times\#L - c \le q-p$.

Let S , be specifications, defined informally as follows.

S = (p',..q' is a longest palindromic segment) $\wedge$ $t' \le t + (\#L)^2/4$

R = (p,..q is a longest palindromic segment with center $\le c/2$) $\Rightarrow S$

Q = (i',..j' is the longest palindromic segment with center $c/2$) $\wedge$ $c'=c \wedge p'=p \wedge q'=q$

P = (i,..j is a palindromic segment with center $c/2$) $\Rightarrow Q$

Here are the refinements.

$S \Leftarrow c := 0.\ p := 0.\ q := 0.\ R$

$R \Leftarrow$ **if** $2 \times \#L - c \leq q - p$ **then** *ok*
else $Q.$ **if** $j - i > q - p$ **then** $p := i.\ q := j$ **else** *ok* **fi**. $c := c+1.\ R$ **fi**

$Q \Leftarrow$ **if** *even c* **then** $i := c/2.\ j := c/2$ **else** $i := (c-1)/2.\ j := (c+1)/2$ **fi**. P

$P \Leftarrow$ **if** $i = 0 \vee j = \#L$ **then** *ok* **else if** $L\ (i-1) \neq L\ j$ **then** *ok* **else** $i := i-1.\ j := j+1.\ P$ **fi fi**

I haven't done the timing and I haven't done the proofs. Sorry.