|  |  |  |
| --- | --- | --- |
| int palindrom(char \*word) { | najgori | najbolji |
|  int i, j, n; |  |  |
|  n = strlen(word); |  |  |
|  i = 0; |  |  |
|  j = n-1; |  |  |
|  while (i < j) { | n/2+1 | 1 |
|  if (word[i] != word[j]) { | n/2 | 1 |
|  return false; |  |  |
|  } |  |  |
|  else { |  |  |
|  ++i; --j; |  |  |
|  } |  |  |
|  } |  |  |
|  return true; |  |  |
| } |  |  |

$f\_{1}$ – broj instrukcija u najgorem slučaju, $f\_{2}$ – broj instrukcija u najboljem slučaju, $g$ – broj zauzetih memorijskih jedinica, broj karaktera

$$f\_{1}(n)=\frac{n}{2}+1+\frac{n}{2}=n+1=O(n)$$

$$f\_{2}(n)=1+1=2=Ω(1)$$

$$g\left(n\right)=n=θ(n)$$

|  |  |  |
| --- | --- | --- |
| int palindrom(char \*word, int n) { |  | n |
|  int i; |  |  |
|  if (n<2) { |  |  |
|  return true;  |  |  |
|  } |  |  |
|  if (word[0] != word[n-1]) { |  |  |
|  return false; |  |  |
|  } |  |  |
|  char \*new\_word = (char \*)malloc((n-1)\*sizeof(char)); |  | (n-2)+(n-4)+…+2 |
|  for (i=0; i<n-2; i++) { |  |  |
|  new\_word[i] = word[i+1]; | (n-2)+(n-4)+…+2 |  |
|  } |  |  |
|  new\_word[n-2] = '\0'; |  |  |
|  return palindrom(new\_word, n-2); |  |  |
| } |  |  |

$$f\left(n\right)=\left(n-2\right)+\left(n-4\right)+…+2= 2\left(1+2+…+\frac{n}{2}-1\right)=2\frac{\frac{n}{2}∙\left(\frac{n}{2}-1\right)}{2}=\frac{n^{2}}{4}-\frac{n}{2}=O(n^{2})$$

$$g\left(n\right)=\frac{n^{2}}{4}-\frac{n}{2}=O(n^{2})$$

Prokomentarisati i druge aspekte koji utiču na efikasnost algoritma. Tu se pre svega misli na:

 - vreme za instanciranje funkcija

 - vreme za alokaciju memorije

 - alocirana memorija se nigde ne oslobađa, da li će da primete taj problem.

Svaku stavku koju uoče bodovati nekim brojem poena kao bonus.