|  |  |  |
| --- | --- | --- |
| 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; |  |  |
| } |  |  |

– broj instrukcija u najgorem slučaju, – broj instrukcija u najboljem slučaju, – broj zauzetih memorijskih jedinica, broj karaktera

|  |  |  |
| --- | --- | --- |
| 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); |  |  |
| } |  |  |

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.