RAČUNARSKA GIMNAZIJA

MATURSKI RAD
iz predmeta
Napredne tehnike programiranja

Vizualizacija algoritama za sortiranje

Učenik
Lav Leon Hudak, IV$_1$

Mentor
dr Filip Marić

Beograd, jun 2019.

Sadržaj

Tehnologije korišćene u projektu

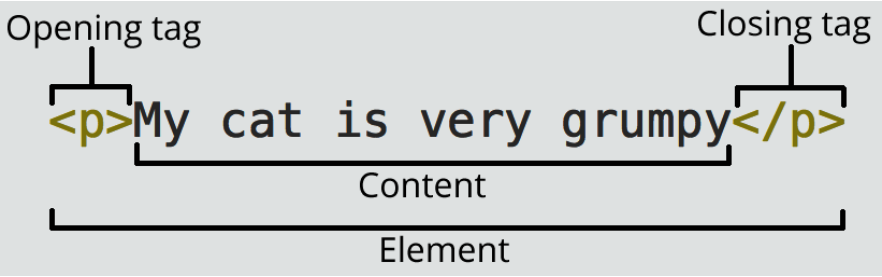
HTML

Hypertext Markup Language (HTML) je programski jezik koji se koristi za kreiranje veb stranica. Često se koristi u tandemu sa drugim jezicima, kao što su Cascading Stylesheets (CSS), JavaScript (JS) i PHP: Hypertext Processor (PHP).

HTML dokument je zapravo običan tekst fajl. Kada internet pretraživač otvori ovaj falj, on traži HTML kod i koristi ga da prikaže paragrafe, ubaci slike, napravi strukturu veb stranice itd. HTML fajl ima nekoliko osnovnih elemenata: *<html>, <head>* i *<body>*. Na primeru (1) može se primetiti da postoje i *</html>*, *</head>* i *</body>* tag-ovi. Ovo je zato što se skoro svi HTML elementi sastoje iz tag-a koji ih otvara, i tag-a koji ih zatvara. Takođe, može se videti da u fajlu postoji takozvana „parent-child“ struktura; elementi se stavljaju jedni unutar drugih.

```
1   <!DOCTYPE html>
2   <html>
3   <head>
4     <title>Page Title</title>
5   </head>
6   <body>
7     <h1>This is a Heading</h1>
8     <p>This is a paragraph.</p>
9   </body>
10  </html>
```

(1) Primer osnovnog HTML koda



(2) Struktura HTML elementa

CSS

Cascading Stylesheets (CSS) se koristi da „ulepšamo“ veb stranicu. Ako HTML daje strukturu veb stranici, CSS obezbeđuje pravila po kojima će se HTML elementi prikazivati. Možemo im menjati boju, veličinu, poziciju i mnogo više.

```css
p {
  text-decoration: none;
  font-family: 'Helvetica', 'Arial', sans-serif;
  font-size: 20px;
}
```

(3) Primer CSS koda

JavaScript

JavaScript (JS) je skripting jezik koji se koristi najviše kao dodatak veb stranicama, ali takodje ima upotrebu u drugim okruženjima kao što su Node.js i Apache. Koristeći JavaScript, implementiramo stvari kao što su animacije, 2D i 3D grafike, interaktivne mape i ostale dinamičke delove veb stranice. JavaScript ima takozvanu „curly-bracket“ sintaksu, i spada u programske jezike visokog nivoa, što znači da je njegova sintaksa lako čitljiva za ljude.

Kako bismo uključili JavaScript fajl u HTML stranicu, potrebno je dodati joj *<script>* tag, i dati mu atribut *src*. Ovom atributu prosleđujemo put do našeg JavaScript fajla, koji se završava ekstenzijom *.js*. *<script>* tag se može staviti bilo gde u stranici, ali pošto se stranica učitava odozgo na dole, kako bi se svi HTML elementi učitali pre JS koda, dobra je praksa da se ovaj element stavi na sam kraj stranice.

```javascript
const para = document.querySelector('p');

para.addEventListener('click', updateName);

function updateName() {
  let name = prompt('Enter a new name');
  para.textContent = 'Player 1: ' + name;
}
```

(4) Primer JavaScript koda

Processing

Processing je Java biblioteka otvorenog koda i okruženje koja omogućava pravljenje digitalne umetnosti, animacije i video igrica sa ciljem učenja ljudi o programiranju u vizuelnom kontekstu. Processing se prvi put pojavio 2001. godine, kada su se Java Applet-i koristili za prikazivanje grafičkih radova na internetu. Ovaj programski okvir koristi takozvani „Sketchbook“, na kome se iscrtava grafika. Pored mnogih funkcija koje omogućavaju lak grafički prikaz, Processing ima dve ključne funkcije:

1. **Setup** je funkcija koja se pokrene čim program započne sa radom. U njoj se definišu početne osobine programa, kao npr. veličina prozora u kome će se prikazivati. Svaki program može imati samo jednu setup() funkciju, i nju ne bi trebalo manualno pozivati.
2. **Draw**, sa druge strane, je funkcija koja se izvršava beskonačno mnogo puta, dok se program ne zaustavi ili se pozove funkcija noLoop(). Poziva se automatski i posle njenog izvršavanja sadržina programa se ponovo iscrta na ekranu.

```
void setup() {
  size(640, 360);
  noStroke();
  rectMode(CENTER);
}

void draw() {
  background(51);
  fill(255, 204);
  rect(mouseX, height / 2, mouseY / 2 + 10, mouseY / 2 + 10);
  fill(255, 204);
  int inverseX = width - mouseX;
  int inverseY = height - mouseY;
  rect(inverseX, height / 2, (inverseY / 2) + 10, (inverseY / 2) + 10);
}
```

(5) Primer Processing koda

Processing ima širok spektar funkcionalnosti, kao što su:

- praćenje događaja kao što su *mousePressed* i *keyPressed*,
- crtanje fundamentalnih oblika kao što su kvadrati, trouglovi i poligoni,
- odeljak za matematiku; funkcije kao što su sinus i kosinus, interpolacija i sl,
- iscrtavanje Bezjeovih krivih, korišćenje Perlinovog šuma,
- pretvaranje polarnih koordinata u koordinate u Dekartovom koordinatnom sistemu,
- i mnogo više.

p5.js

p5.js je JavaScript biblioteka napravljena na osnovu Processing-a. Napravljena je sa istim ciljem kao i Processing, tj. da približi svet programiranja dizajnerima, umetnicima i ljudima koji žele novi način da se kreativno izraze. p5.js koristi element *<canvas>* kao prostor na kome se iscrtava grafika, nalik Sketchbook-u u Processing-u. Ova biblioteka sadrži mnoge funkcije koje olakšavaju grafički rad, a neke od kategorija funkcija su:

- Color - sadrži funkcije za rad sa bojama, kao što je menjanje režima rada iz RGB u HSB,
- Shape - sadrži funkcije za crtanje primitivnih oblika, kao što su *circle()*, *rect()* i *point()*,
- Math - funkcije koje se bave vektorima i njihovim osobinama,
- Events - skup funkcija za rad sa događajima, kao što su *mouseMoved* i *keyPressed*.

p5.js takodje olakšava HTML aspekat programiranja, jer sadrži odeljak koji se bavi DOM-om. U ovom odeljku postoje funkcije koje se bave direktnim kreiranjem HTML elemenata iz JavaScripta. Primer ovakve funkcije je *createButton().* Takođe, postoje funkcije kao što su *child()* i *parent()*, koje omogućavaju pozicioniranje elemenata unutar HTML strukture.

Ako želimo da koristimo p5.js biblioteku na našoj stranici, postoje dva načina:

1. možemo da uključimo p5.js CDN link u naš *<script>* element, ili
2. možemo da skinemo celu biblioteku (koja nije velika) u direktorijum našeg projekta.

```
let y = 100;

function setup() {
  createCanvas(720, 400); // Veličina kanvasa se mora navesti prva
  stroke(255); // Boja crtanja se menja u belu
  frameRate(30);
}
// Kod u draw() se izvršava dok se progaram ne zaustavi
function draw() {
  background(0); // Boja pozadine se menja u crnu
  y = y - 1;
  if (y < 0) {
    y = height;
  }
  line(0, y, width, y); // iscrtava se linija koja ide nagore
}
```

(6) Primer p5.js koda

Vizualizacija algoritama za sortiranje

Cilj projekta

Ako pogledamo oko sebe, možemo primetiti da smo okruženi nizovima različitih vrsta. Knjige na policama, kalendari, bilo kakvi natpisi, pa čak i slova na tastaturi. Ako ste ikada igrali tablićе ili poker, postoji šansa da ste karte u ruci poređali po boji, vrednosti ili nekom drugom redosledu, kako bi njihov odabir bio lakši. Kada želimo da rezervišemo sobu u hotelu, sajtovi nam pružaju opcije da prikažemo prvo one najjeftinije, najbliže centru ili sa najboljom ocenom. Još jedan primer su knjige u biblioteci, takođe poređane, i to leksikografski - po abecedi ili azbuci.

U svetu programiranja nizovi su vrlo bitan koncept; u njima se čuvaju svi podaci koje vidimo (a i oni koje ne vidimo). Nizovi su zapravo promenljiva u kojoj čuvamo više promenljivih. Promenljive koje su unutar niza obično nazivamo „elementima niza“. Svaki element ima svoju poziciju u nizu, a u većini programskih jezika brojanje pozicije počinje od nule. Na ovo treba pripaziti kada radimo sa nizovima.

Nalaženje elemenata jeste jedna od najvažnijih operacija koju možemo da izvršimo na jednom nizu. Kako bismo ubrzali nalaženje pojedinačnog elementa, skoro uvek pre algoritma za traženje koristimo algoritam za sortiranje niza. Postoji puno ovakvih algoritama: neki su brži, neki sporiji, neki više prikladni za neke situacije od drugih. Postoje i oni koji se implementiraju rekurzivno. Kroz ovaj rad videćemo kako se implementiraju, i kako izgledaju tri prosta algoritma za sortiranje: Bubble Sort, Insertion Sort i Selection Sort.

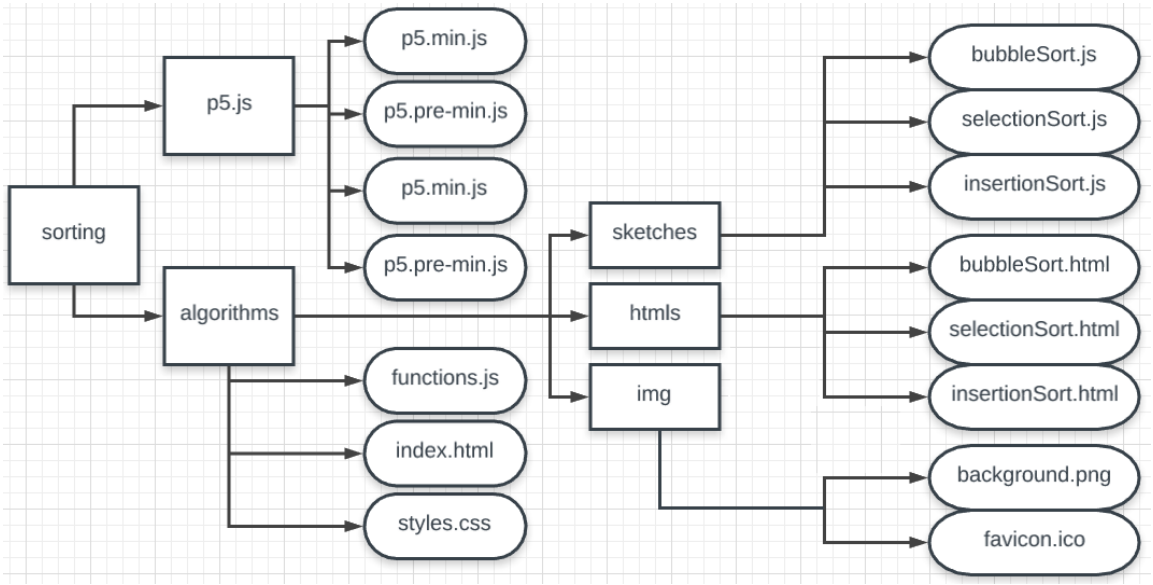
Struktura projekta

Svi fajlovi (izuzev fajlova p5.js biblioteke) se nalaze u folderu *algorithms*. Glavna (početna) stranica projekta, *index.html*, sadži kratak uvod o tome šta su to nizovi u programiranju, par sintaksičkih primera, i kartice koje vode na ostale stranice. Ova stranica, kao i ostale HTML stranice, dele jedan CSS fajl, *styles.css*. Nakon toga, imamo tri direktorijuma:

- htmls,
- sketches,
- img.

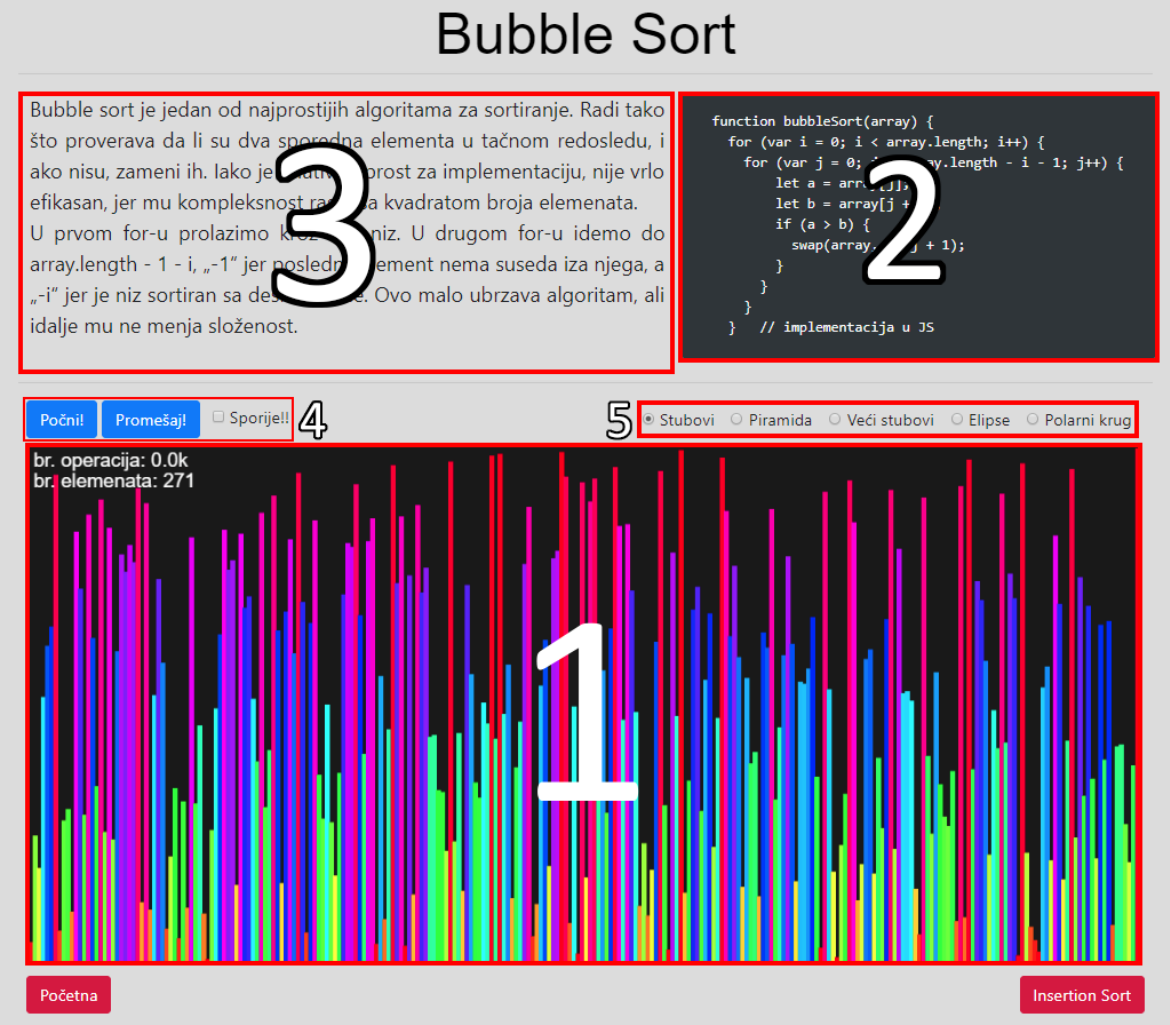
U direktorijumu *htmls* se nalaze HTML stranice za pojedinačne algoritme, naime Bubble Sort, Insertion Sort i Selection Sort. U direktorijumu *img* nalaze se pozadina, koju sve stranice koriste, i ikonica za sajt, a u direktorijumu *sketches* nalaze se pojedinačni JS fajlovi: *bubbleSort.js*, *insertionSort.js* i *selectionSort.js*. Pošto stranice koje prikazuju algoritme međusobno koriste dosta istog JS koda, ideja je da svaka od tih stranica sadrži fajlove *functions.js* (u kome se nalaze zajedničke funkcije i promenljive) i *xzySort.js* (koji implementira unikatni algoritam). Cilj ovoga je da se ponavljanje koda smanji na minimum, što olakšava navigaciju kroz sam kod.

U ovom projektu korišćen je popularni sistem za kontrolu verzija, *Git*, radi lakšeg premeštanja i očuvanja koda i njegove istorije. Git je povezan sa sajtom GitHub koji, uz nalog, daje veliku funkcionalnost za menadžment projekta, kao što su push i pull zahtevi, nove grane, i onlajn prikaz koda.



(7) Struktura projekta

Objašnjenje HTML stranice



(8) Prikaz stranice bubbleSort.html

1. Kanvas na kome je prikazan izmešani niz i informacije o radu algoritma
2. Kod algoritma koji sortira niz
3. Objašnjenje algoritma i njegovog koda
4. Dugmići za početak/zaustavljanje sortiranja, dugme za ponovno mešanje niza, checkBox koji omogućava sporije sortiranje niza
5. Dugmići za odabir načina iscrtavanja; trenutko prikazani *stubovi*

Sve stranice koje su zadužene za pojedine algoritme imaju isti raspored, ali drugačiji sadržaj.

functions.js

U fajlu functions.js je kod koji dele sve tri stranice koje prikazuju algoritme. U njemu se nalaze:

1. deklaracije globalnih promenljivih,
2. funkcija za resetovanje kanvasa i popunjavanje niza,
3. funkcije za rad sa nizom, kao što su *swap* i *shuffleArray,*
4. funkcije koje obrađuju događaje sa HTML stranice,
5. funkcije koje služe za iscrtavanje na kanvasu.

Globalne promenljive

Većina promenljivih korišćenih u ovom projektu su zajedničke za sve algoritme:

```
let niz; // sadrži niz
let n; // broj elemenata u nizu
let maxNiza; // vrednost najvećeg elementa u nizu
const rectWidth = 4;
let ssRectWidth = rectWidth * 8;
let u; // globalni brojač koji se koristi u draw()
let numOps; // broj operacija
let canvas; // promenljiva u kojoj se čuva kanvas
let step; // razmak između dva elementa u nizu
let firstLoop = true; // označava prvo iscrtavanje na kanvasu
```

Takođe, imamo boolean promenljive koje određuju kako će se algoritam iscrtavati:

```
let stubovi = false;
let piramida = false;
let veciStubovi = false;
let elipse = false;
let polarCircle = false;
```

Ove promenljive se koriste i za pravljenje niza, jer je za svako iscrtavanje potreban malo drugačiji niz. Ovo se dešava u funkciji *resetSketch()*.

U sledećem bloku koda uzimamo podatke o HTML elementima na stranici, kao što su dugmići, checkBox-ovi i radioButton-i. Takođe uzimamo podatke kao što su širina i visina * * elementa u kome se nalazi kanvas, na osnovu kojih kasnije određujemo veličinu i poziciju samog kanvasa.

```
let startStopBtn = document.getElementById('startStopBtn');
let resetBtn = document.getElementById('resetBtn');
let rbStubovi = document.getElementById('rbStubovi');
let rbPiramida = document.getElementById('rbPiramida');
let rbVStubovi = document.getElementById('rbVStubovi');
let rbElipse = document.getElementById('rbElipse');
let fpsChanger = document.getElementById('fpsChanger');

var containerDiv = document.getElementById('glavni');
var positionInfo = containerDiv.getBoundingClientRect();
var dHeight = positionInfo.height;
var dWidth = positionInfo.width;
```

resetSketch()

Ova funkcija se poziva svaki put kada bi canvas trebalo da se resetuje. Ove situacije su:

1. kada se pozove funkcija *setup()* - zato što je u *resetSketch()* funkciju prebačen kod koji se izvršava kada se prvi put niz pravi i izcrtava na kanvasu,
2. kada korisnik klikne dugme „Promešaj!“,
3. kada se veličina prozora promeni,
4. u funkciji *rbChanged()*, koja je obrađivač događaja za radioButton-e koji menjaju način iscrtavanja na kanvasu.

resetSketch() ima posao da vrati sve na početno stanje, a potom proveri kakvo iscrtavanje je izabrano, i napravi niz u odnosu na to.

```
function resetSketch() {
  u = 0;
  numOps = 0;
  firstLoop = true;
  // proveravanje radioButton-a za iscrtavanje
  if (rbStubovi.checked) {
    stubovi = true;
  } else if (rbPiramida.checked) {
    piramida = true;
  } else if (rbVStubovi.checked) {
    veciStubovi = true;
  } else if (rbElipse.checked) {
    elipse = true;
  } else if (rbPolarCircle.checked) {
    polarCircle = true;
  }
```

Funkcija prvo vraća globalni brojač *u* i broj operacija na vrednost 0, a zatim vrednost promenljive *firstLoop* na true. Nakon toga, proverava koji način iscrtavanja je izabran, i na osnovu toga daje vrednost true jednoj od promenljivih u odnosu na koju se kasnije pravi niz.

Sledeći deo koda bavi se određivanjem broja elemenata i njihovih vrednosti u nizu. Niz koji nastaje je aritmetički niz, u kome je razlika između dva elementa vrednost promenljive *step*. p5.js daje promenljive *width* i *height,* čije su vrednosti širina i visina kanvasa.

```
if (piramida) {
  n = ceil(width);
  niz = new Array(n);
  step = (height / niz.length) / 2;
} else if (stubovi) {
  n = ceil(width / rectWidth);
  niz = new Array(n);
  step = (height / niz.length);
} else if (elipse) {
  n = ceil(2 * width / 5);
  niz = new Array(n);
  step = 1;
} else if (veciStubovi) {
  n = ceil(width / ssRectWidth);
  niz = new Array(n);
  step = (height / niz.length);
} else if (polarCircle) {
  n = 360;
  niz = new Array(n);
  step = 1;
}

niz[0] = step;
for (var i = 1; i < niz.length; i++) {
  niz[i] = niz[i - 1] + step;
}
maxNiza = niz[niz.length - 1];
colorMode(HSB, maxNiza);
shuffleArray(niz);
```

U slučaju **piramide** imamo elemenata koliko i piksela po širini kanvasa. Svaka linija je zapravo element. Visinu, tj. polovinu visine ove linije daje nam vrednost elementa. Kako bismo imali linije od sredine leve ivice do gornjeg i donjeg desnog ugla kanvasa, potrebno je da *step* bude 1/2 količnika visine kanvasa i dužine niza.[1]

U slučaju **stubova** i **većih stubova**, broj elemenata se računa kao *ceo*[2] broj najbliži odnosu širine kanvasa i stubova koji se iscrtavaju. Kako bismo imali „liniju" od donjeg levog ugla do gornjeg desnog ugla kanvasa, *step* se računa kao količnik visine kanvasa i dužine niza.

Za **elipsu** su potrebne vrednosti velike i male poluose; njih dobijamo preko pozicije elementa u nizu. Ovo znači da će širina elipse biti 4/5 širine kanvasa, a pola od toga.

Za **polarni krug i**mamo 360 elemenata, i fiksiranu vrednost promenljive *step.*

Potom popunimo niz prostim *for* ciklusom, i sačuvamo najveću vrednost u nizu pre nego što ga promešamo.

Kako bismo prikazali spektar boja, koristimo funkciju *colorMode()* da izaberemo režim rada HSB („Hue, saturation, brightness"), kojoj kao drugi agrument treba proslediti *range,* tj. maksimalnu vrednost. Da bismo prikazali ceo spektar u tačnom odnosu sa načinom iscrtavanja, ovde prosleđujemo promenljivu *maxNiza.*

[1] Detaljnije objašnjeno iscrtavanje svih oblika u odelku „Funkcije za iscrtavanje na kanvasu".
[2] Broj elemenata u nizu mora biti ceo broj.

Funkcije za rad sa nizom

Imamo dve funkcije za rad sa nizovma, *swap()* i *shuffleArray()*.

```
function swap(array, a, b) {
  let temp = array[a];
  array[a] = array[b];
  array[b] = temp;
}

function shuffleArray(array) {
  var currentIndex = array.length;
  var randomIndex;
  for (var i = array.length; i > 0; i--) {
    randomIndex = Math.floor(Math.random() * currentIndex);
    currentIndex -= 1;
    swap(array, currentIndex, randomIndex);
  }
}
```

swap() uzima kao argumente niz na kome radi i pozicije elemenata koje treba da zameni. Potom sprovodi prostu zamenu elemenata, koristeći privremenu promenljivu.

shuffleArray() uzima niz koji treba da izmeša kao argument. Kao početnu vrednost uzimamo broj elemenata u nizu. Kako bismo dobili neku nasumičnu poziciju u nizu, označenu sa *randomIndex*, uzimamo vrednost *currentIndex* i množimo je sa Math.random(), koja vraća brojeve od 0 do 1, isključujući 1, a zatim je zaokružujemo na najbliži ceo broj ispod. Potom oduzimamo jedan od *currentIndex* i pozivamo funkciju *swap()*, koja zamenjuje elemente na poziciji *currentIndex* i *randomIndex*. Ovo se ponavlja onoliko puta koliko imamo elemenata u nizu, kako bismo bili sigurni da je ceo niz promešan.

Funkcije koje obrađuju događaje sa HTML stranice

```
function rbChanged() {
  stubovi = false;
  piramida = false;
  veciStubovi = false;
  elipse = false;
  polarCircle = false;
  resetSketch();
  fpsChanged();
}
```

```
function fpsChanged() {
  if (fpsChanger.checked) {
    frameRate(2);
  } else {
    frameRate(60);
  }
}
```

Funkcija *rbChanged()* je obrađivač događaja za sve radioButton-e koje su na stranici. Ova funkcija resetuje sve boolean vrednosti za crtanje na false, poziva *resetSketch()* (u kojoj se proverava koji način iscrtavanja je izabran), i poziva funkciju *fpsChanged()*, koja je zadužena da obradi događaj od checkBox-a koji menja broj kadrova koji se iscrtavaju na kanvasu po sekundi.

```
function windowResized() {
  positionInfo = containerDiv.getBoundingClientRect();
  dHeight = positionInfo.height;
  dWidth = positionInfo.width;
  resizeCanvas(9.5 * dWidth / 10, 500);
  resetSketch();
}
```

Funkcija *windowResized()* je ugrađena u p5.js biblioteku i poziva se kada se veličina prozora u kome se kanvas prikazuje promeni. Uzima se nova visina i širina * * elementa u kome se nalazi kanvas, poziva se ugrađena funkcija *resizeCanvas()* koja menja širinu kanvasa na 95% * * elementa u kome je, a potom se poziva funkcija *resetSketch()*.

```
function startStopSketch() {
  // start/stop dugme onClick
  firstLoop = false;
  if (startStopBtn.innerHTML == "Počni!") {
    loop();
    startStopBtn.innerHTML = "Stani!";
  } else {
    noLoop();
    startStopBtn.innerHTML = "Počni!";
  }
}
```

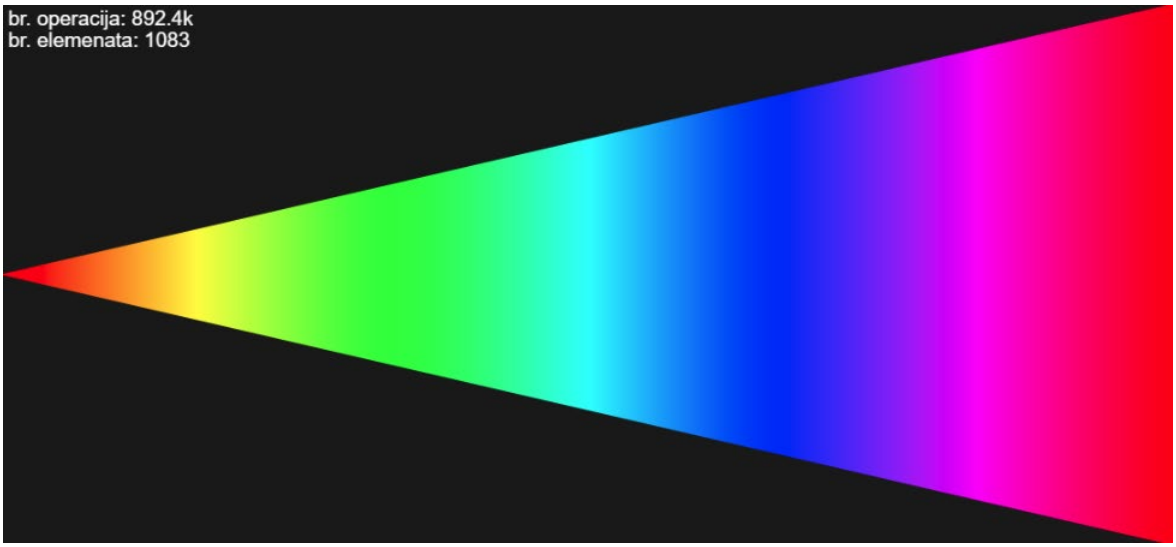
Funkcija *startStopSketch()* zadužena je da pauzira i pokrene iscrtavanje na kanvasu. Ona se poziva na događaj *onClick* dugmeta *startStopBtn*. Ovde koristimo funkcije *loop()* i *noLoop()*, koje su ugrađene u p5.js biblioteku. Ovo su funkcije koje zapravo prekidaju u ponovo pokreću iscrtavanje na kanvasu. Pored toga, menjamo vrednost firstLoop na false (jer više nije prvo iscrtavanje) i menjamo tekst na dugmetu kako bi se uklapao sa stanjem kanvasa.

Funkcije za iscrtavanje na kanvasu

Funkcija *crtanje()* ima *if else* slučajeve od kojih se svaki bavi iscrtavanjem posebnog oblika. To su:

1. piramida,
2. stubovi i veći stubovi,
3. elipse,
4. polarni krug.

Piramida

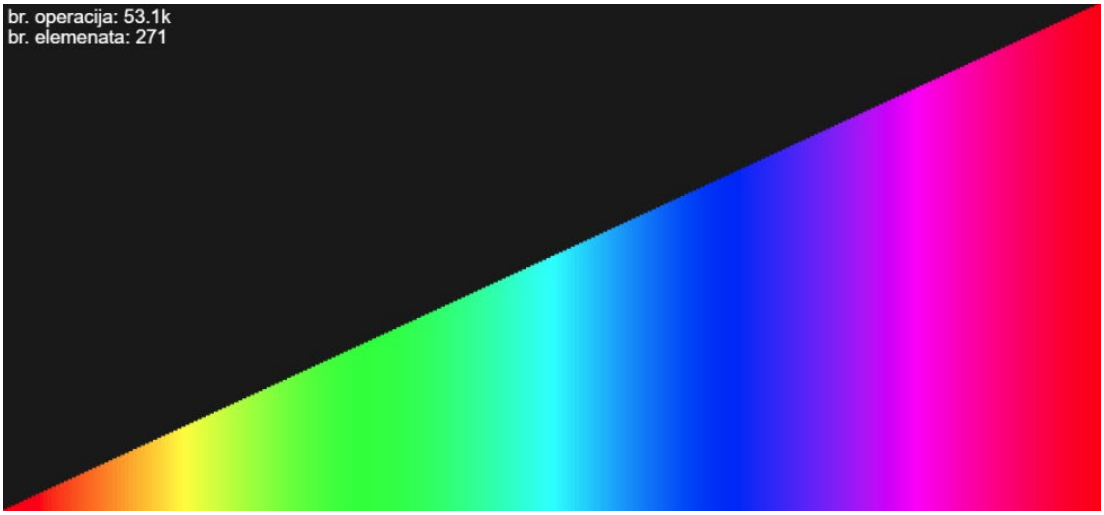


Kako bismo iscrtali spektar boja, koristimo funkciju *stroke()* koja menja boju ivice crtanja. *stroke()* se može pozivati sa različitim brojem argumenata, naime jedan, tri ili četiri. U našem slučaju, poziva se sa tri argumenta. Ako bismo bili u RGB režimu rada, ovo bi bili parametri *red*, *green* i *blue,* ali pošto koristimo HSB režim rada, ovo su *hue*, *saturation* i *brightness*. Pošto smo u funkciji *resetSketch()* prosledili vrednost maxNiza kao maksimalne vrednosti ova tri parametra, za *brightness* i *saturation* prosleđujemo maksimalnu vrednost, a za hue vrednost niza u tom trenutku, pošto nam je niz artitmetički i ravnomerno raspoređen. Ovaj način razmišljanja je isti za sve načine crtanja.

```
function crtaj() {
  if (piramida) {
    for (var i = 0; i < niz.length; i++) {
      stroke(niz[i], maxNiza, maxNiza);
      line(i, height / 2, i, height / 2 - niz[i]);
      line(i, height / 2, i, height / 2 + niz[i]);
    }
  }
```

Kod piramide, svaka linija (dve linije) predstavlja jedan element. Piramida ima osu simetrije po liniji *height / 2*. Funkcija kojom crtamo linije, *line()*, uzima početnu tačku i krajnju tačku, tj. njihove koordinate. Prva tačka je ista za obe linije, i njena x koordinata je u stvari pozicija trenutnog elementa u nizu, a y koordinata se nalazi na osi simetrije. Druga tačka je na istoj vertikali kao i prva, ali joj je y koordinata pomerena za vrednost niza u pozitivnom ili negativnom smeru.

Stubovi i veći stubovi

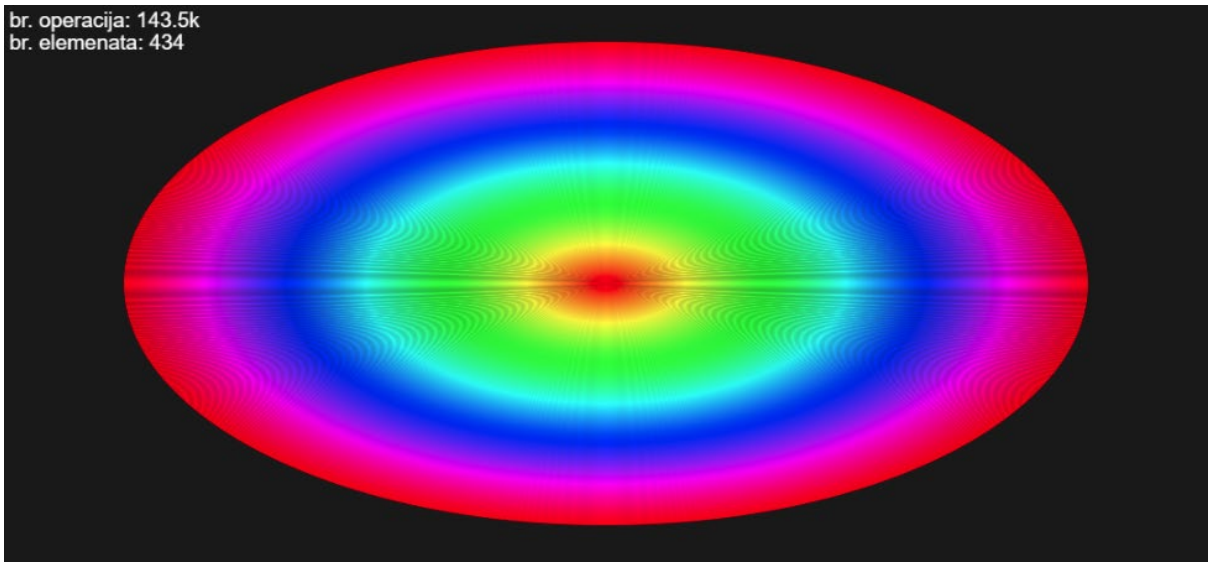


Jedina razlika kod stubova i većih stubova jeste zapravo širina stubova. Širina stubova je određena promenljivom *rectWidth*, a širina većih stubova je samo umnožak ove promenljive.

```
else if (stubovi) {
 for (var i = 0; i < niz.length; i++) {
   fill(niz[i], maxNiza, maxNiza);
   stroke(niz[i], maxNiza, maxNiza);
   rect(i * rectWidth, height - niz[i], rectWidth, niz[i]);
 }
} else if (veciStubovi) {
 for (var i = 0; i < niz.length; i++) {
   stroke(niz[i], maxNiza, maxNiza);
   fill(niz[i], maxNiza, maxNiza);
   rect(i * ssRectWidth, height - niz[i], ssRectWidth, niz[i]);
 }
}
```

Funkcija *rect()* uzima gornju levu tačku, visinu i širinu pravougaonika koji crta. Pošto su stubovi jedan do drugog, *x* koordinata gornje leve tačke će biti proizvod širine stuba i njegove pozicije u nizu, a *y* koordinata se nalazi na poziciji *height* minus vrednost trenutnog elementa. Širina stuba je opet, promenljiva *rectWidth*, a visina stuba je vrednost elementa u tom trenutku. Sistem crtanja većih stubova je poptuno isti.

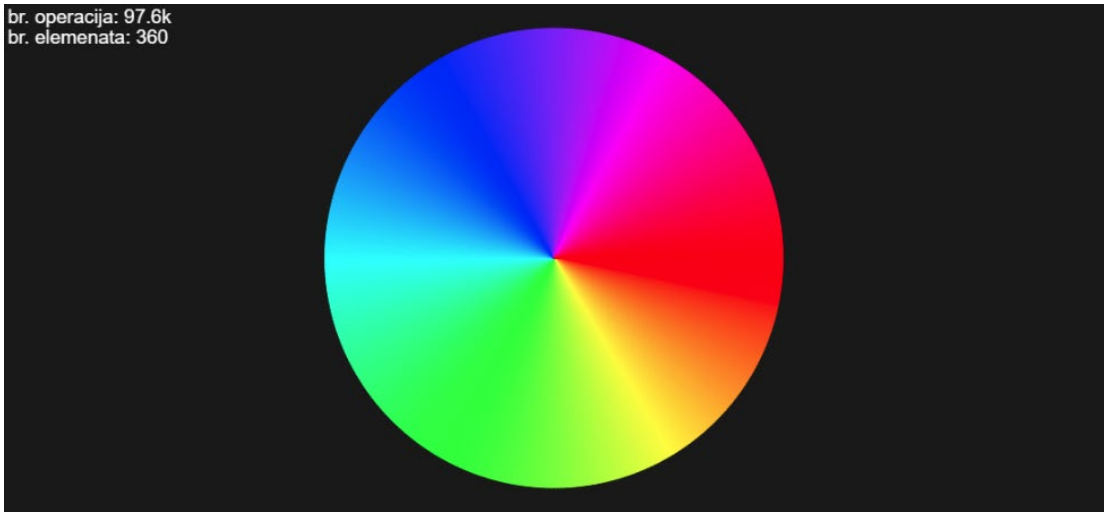
Elipse



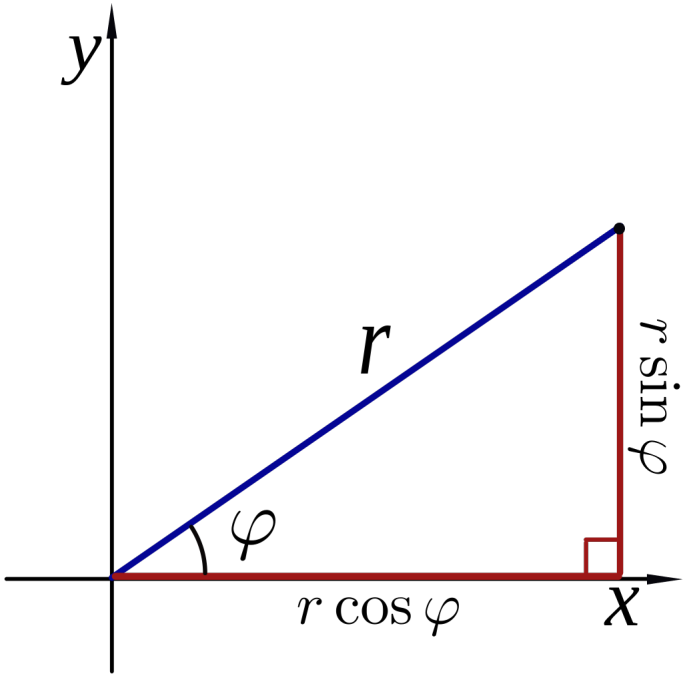
Ideja kod ovog prikaza jeste da se za svaki element nacrta jedna elipsa, koja nije popunjena. Kako elipse ne bi bile popunjene, koristimo p5.js funkciju *noFill()*. Zatim biramo boju za trenutnu elipsu, i crtamo je sa p5.js funkcijom *ellipse()*. Ova funkcija uzima kao parametre centar elipse, njenu visinu i njenu širinu. Pošto se centar elipse nalazi u centru kanvasa, prosleđujemo joj koordinate centra *width / 2* i *height / 2*. Za širinu joj dajemo trenutnu poziciju elementa u nizu, a za visinu pola od toga.

```
else if (elipse) {
  noFill();
  for (var i = 0; i < niz.length; i++) {
    stroke(niz[i], maxNiza, maxNiza);
    ellipse(width / 2, height / 2, i, i / 2);
  }
}
```

Polarni krug



Za svaki element u nizu crta se poluprečnik ovog kruga. Pošto su elementi u nizu celi brojevi od 1 do 360, pozicija elementa u nizu je zapravo i ugao pod kojim će on biti crtan. Kako bismo odredili koordinate za crtanje, koristimo sistem polarnih koordinata, koji je objašnjen ispod.



(1) Sistem polarnih koordinata

Svaka tačka na kružnici je određena uglom φ i razdaljinom od centra kruga *r*. *X* koordinata je data jednačinom *r * cos(φ)*, a *y* koordinata jednačinom *r * sin(φ)*.

Pošto funkcije Math.sin i Math.cos uzimaju ugao u radijanima, koristimo formulu navedenu dole da u promenljivu *theta* sačuvamo trenutni ugao pod kojim crtamo *trougao* (umesto linije). Crtamo trougao zbog lepšeg prikaza. Tačke na kružnici koje trougao sadrži su tačke koje su pomerene za pola stepena unazad, tj. unapred od ugla pod kojim se iscrtava trenutni element. Funkcija *triangle()* za argumente uzima koordinate sve tri tačke, a *fill()* i *stroke()* boje ovaj trougao.

```
    else if (polarCircle) {
      let r = height * 0.45;
      let x0 = width / 2;
      let y0 = height / 2;
      let halfDeg = 0.5 * (Math.PI / 180);
      for (var i = 0; i < niz.length; i++) {
        let theta = i * (Math.PI / 180);
        let xl = x0 + r * Math.cos(theta - halfDeg);
        let yl = y0 + r * Math.sin(theta - halfDeg);
        let xr = x0 + r * Math.cos(theta + halfDeg);
        let yr = y0 + r * Math.sin(theta + halfDeg);
        fill(niz[i], maxNiza, maxNiza);
        stroke(niz[i], maxNiza, maxNiza);
        triangle(x0, y0, xl, yl, xr, yr);
      }
    }
}
```

Funkcija za ispis podataka o algoritmu

```
function ispisiPodatke() {
  noStroke();
  colorMode(RGB);
  fill(255);
  let brUtxt = "br. operacija: " + (numOps / 1000).toFixed(1) + "k";
  let nTxt = "br. elemenata: " + n;
  textSize(19);
  text(brUtxt, 5, 20);
  text(nTxt, 5, 40);
  colorMode(HSB, maxNiza);
}
```

U funkciji *ispisiPodatke()* u levom gornjem uglu kanvasa ispisujemo podatke o broju elemenata u nizu koji se trenutno prikazuje, i broj operacija koje je algoritam izvršio do datog trenutka. Moramo promeniti režim rada u RGB jer se u odnosu na način iscrtavanja menja *range*, tj. maksimalna vrednost, pa ne bismo mogli da imamo belu boju sve vreme. Broj operacija se ispisuje u hiljadama tako što podelimo pravi broj operacija sa hiljadu i odsečemo sve osim poslednje decimale broja kojeg smo dobili sa funkcijom *toFixed()*. Nakon što ispišemo sve podatke, vraćamo režim rada na HSB.

bubbleSort.js

bubbleSort.js je prvi od tri fajla za pojedinačne algoritme. On implementira prost algoritam za sortiranje, Bubble Sort. Bubble sort ima prosečnu složenost $O(n^2)$, i zato nije efikasan za sortiranje velikih nizova.

```
function bubbleSort(array) {
  for (var i = 0; i < array.length; i++) {
    for (var j = 0; j < array.length - i - 1; j++) {
      let a = array[j];
      let b = array[j + 1];
      if (a > b) {
        swap(array, j, j + 1);
      }
    }
  }
}
```

(9) Kod Bubble Sort algoritma

Svaki fajl koji implementira algoritam ima istu strukturu: funkciju *setup()*, i funkciju *draw()*. Nevezano za to, na dnu svakog fajla postoji celokupan kod algoritma kao primer.

```
function setup() {
  canvas = createCanvas(9.5 * dWidth / 10, 500);
  canvas.parent('sketchContainer');
  resetSketch();
  noLoop();
}
```

Prvo popunjavamo globalnu promenljivu *canvas* sa p5.js funkcijom *createCanvas()*. Zatim stavljamo napravljeni kanvas unutar * * elementa sa id-om *sketchContainer*. Potom pozivamo *resetSketch()*, i zaustavljamo dalje ponavljanje funkcije *draw()*. Ovo radimo jer na početku kanvas treba da bude pauziran, sve dok korisnik ne pritisne dugme za početak rada.

U funkciji *draw()* prvo moramo da postavimo boju pozadine, i ovo radimo sa p5.js funkcijom *background()*. Moramo koristiti isti princip kao kod ispisivanja podataka o algoritmu:

```
function draw() {
  colorMode(RGB);
  background(25);
  colorMode(HSB, maxNiza);
```

Ideja iza animacije je da, pošto u samom algoritmu imamo dva *for* ciklusa, iskoristimo fukciju *draw()*, koja se ponavlja beskonačno mnogo puta, kao spoljašnji ciklus. Postavljamo uslov prvog *for* ciklusa u *if* klauzu, a pritom koristimo globalni brojač *u*. Kao i u pravom *for* ciklusu, na kraju moramo povećati vrednost promenljive *u*. Ostatak koda ostaje isti.

```
if (u < niz.length) {
  for (let j = 0; j < niz.length - 1 - u; j++) {
    var a = niz[j];
    var b = niz[j + 1];
    numOps++;
    if (a > b) {
      numOps++;
      swap(niz, j, j + 1);
    }
  }
}
u++;

if (firstLoop) {
  numOps = 0;
  firstLoop = false;
}
crtaj();
ispisiPodatke();
```

Proveravamo da li je promenljiva *firstLoop* true, i ako jeste, broj operacija stavljamo na 0. Na kraju pozivamo funkcije *crtaj()* i *ispišiPodatke()*. Ovo se ponavlja dok se program ne zaustavi.

insertionSort.js

Ovaj fajl implementira algoritam za sortiranje zvan Insertion Sort. Insertion sort takođe ima prosečnu složenost $O(n^2)$.

```
function insertionSort(array) {
  for (var i = 1; i < array.length; i++)
    for (var j = i; j > 0; j--)
      if (array[j - 1] > array[j])
        swap(array, j, j - 1);
}
```

(10) Kod Insertion Sort algoritma

Prvim for ciklusom prolazimo kroz ceo niz, ali krećemo od pozicije jedan jer nema smisla da podniz koji nastaje ima dužinu 0. Za svaki element na koji „naiđemo“ proveravamo da li je manji od elementa pre njega, i pomeramo ga ako jeste. Kada ne postoji više elemenata koji su manji od trenutnog, našli smo tačnu poziciju za taj element. Nakon ovoga „nailazimo“ na sledeći element, i proces se ponavlja.

Ideja za animaciju je ista kao i kod Bubble Sort-a:

```
if (u + 1 < niz.length) {
  for (var j = u + 1; j > 0; j--) {
    numOps++;
    if (niz[j - 1] > niz[j]) {
      numOps++;
      swap(niz, j, j - 1);
    }
  }
}
u++;
```

Potom se pozivaju funkcije funkcije *crtaj()* i *ispišiPodatke()*.

selectionSort.js

Kao poslednji od tri fajla, *selectionSort.js* implementira Selection Sort. Selection Sort isto ima kvadratnu složenost $O(n^2)$, ali je malo sporiji od Insertion Sort-a jer mora da prolazi kroz ceo nesortiran niz.

```
function selectionSort(array) {
  for (var i = 0; i < array.length - 1; i++) {
    let minValue = array[i];
    let indexOfMin = i;
    for (var j = i + 1; j < array.length; j++) {
      if (minValue > array[j]) {
        minValue = array[j];
        indexOfMin = j;
      }
    }
    if (minValue < array[i])
      swap(array, i, indexOfMin);
  }
}
```

(11) Kod Selection Sort algoritma

Prvim *for* ciklusom prolazimo kroz ceo niz. Uzimamo početnu vrednost *minValue* da bude element na indeksu *i*, a zatim prolazimo kroz podniz čija je prva pozicija *i + 1*. Ako nadjemo element čija je vrednost manja od *minValue*, taj element postaje novi *minValue*. Nakon što prođemo kroz ceo podniz, proverimo da li je *minValue* manji od elementa na poziciji *i*, i ako jeste, zamenjujemo im mesta. Ovaj proces se ponavlja dok se ceo niz ne sortira.

Animacija se ostvaruje na isti način kao kod Bubble i Insertion sort-a.

Zaključak

Kako bismo novim programerima olakšali put koji im predstoji, treba koristiti što više vizualnih primera algoritama. Korišćenjem bilbioteka poput Processing-a i p5.js-a možemo im približiti svet programiranja kroz lep i umetnički način. Ova aplikacija cilja na to da objasni zašto su algoritmi za sortiranje bitni, i umesto da ih samo učimo kroz kod, vidimo i kako oni zapravo izgledaju. Ovom projektu se u budućnosti može dodati još načina prikazivanja, više različitih algoritama za sortiranje, zvuk, i mnogo više.

Kao đak drugog razreda Računarske gimnazije, učio sam algoritme za sortiranje i... nisam ih baš naučio. Od tada sam strahovao da ću se, kao neko ko želi da studira nauku o kompjuterima, mučiti pokušavajući da naučim komplikovanije koncepte u ovoj oblasti nauke. Ovim iako nije previše komplikovanim projektom sam malkice približio sebi oblast algoritama, i oni mi više ne izgledaju toliko zastrašujuće kao pre. Kucajući ovaj kod, dobio sam malo više samouvenja i hrabrosti da nastavim ovim putem.

Reference i literatura

1. Cormen, Thomas H; Leiserson, Charles E; Rivest, Ronald L; Stein, Clifford, Introduction to Algorithms
2. Knuth, Donald E, *The Art of Computer Programming*
3. Marić, Filip, Marinković, Vesna, *Računarstvo i informatika 4*, udžbenik za četvrti razred gimnazije
4. Martin, David R. *Animated Sorting Algorithms*, 2007.
5. *Processing Foundation*, URL: https://processing.org
6. p5.js JavaScript library, URL: https://p5js.org/
7. Shiffman, Daniel, *Nature of Code*, URL: https://natureofcode.com/
8. *Sound of Sorting,* URL: http://panthema.net/2013/sound-of-sorting/