- METAK

```
// MINIMOAREN PROPIETATEA BETETZEN DUEN META
public class Meta {
	private int v[];

	public Meta(int [] bektorea){
		v=bektorea;
		metaEraikiHondoratuz();
		//metaEraikiAzaleratuz();
	}

	public void hondoratu(int indizea){
		int i=indizea;
		int umeMin;
		int aux;
		boolean jarraitu=true;
		while ((2*i+1)<=(v.length-1) && jarraitu){
			umeMin = 2*i+1;
			if ((2*i+2)<=v.length-1 && v[2*i+2]<v[2*i+1]){
				umeMin=2*i+2;
			}
			if(v[i]>v[umeMin]){
				aux=v[i];
				v[i]=v[umeMin];
				v[umeMin]=aux;
				i=umeMin;
			}else{
				jarraitu=false;
			}
		}
	}

	public void azaleratu(int indizea){
		int i=indizea;
		int aux;
		boolean jarraitu=true;
		while (((i+1)/2)-1>=0 && jarraitu){
			if(v[i]<v[((i+1)/2)-1]){
				aux=v[i];
				v[i]=v[((i+1)/2)-1];
				v[((i+1)/2)-1]=aux;
				i=((i+1)/2)-1;
			}else{
				jarraitu= false;
			}
		}
	}

	private void metaEraikiHondoratuz() {
		for(int i=v.length-1;i>=0;i--){
			hondoratu(i);
		}
	}

	private void metaEraikiAzaleratuz() {
		for(int i=0; i<v.length; i++){
			azaleratu(i);
		}
	}

}
```

- BATURA MAXIMOKO SEGMENTUA

```
public class BaturaMaximokoSegmentua {

	public HiruZenbaki baturaMaximokoSegmentua(int [] v, int hasiera, int
amaiera){
		if(hasiera==amaiera){
			HiruZenbaki h = new HiruZenbaki();
			h.hasiera=hasiera;
			h.amaiera=amaiera;
			h.batuketa=v[hasiera];
			return h;
		}else{
			HiruZenbaki h1,h2,h3 = new HiruZenbaki();
			h1 = baturaMaximokoSegmentua(v, hasiera,
(hasiera+amaiera)/2);
			h2 = baturaMaximokoSegmentua(v, ((hasiera+amaiera)/2)+1,
amaiera);
			h3 = baturaMaximokoSegmentuaErdian(v, hasiera,
(hasiera+amaiera)/2, amaiera);
			if ((h2.batuketa <= h1.batuketa) && (h3.batuketa <=
h1.batuketa)) return h1;
			else if ((h1.batuketa <= h2.batuketa) && (h3.batuketa <=
h2.batuketa)) return h2;
			else /*if ((h2.batuketa <= h1.batuketa) && (h3.batuketa <=
h1.batuketa))*/ return h3;
		}
	}

	private HiruZenbaki baturaMaximokoSegmentuaErdian(int[] v, int hasiera,
			int erdia, int amaiera) {
		HiruZenbaki h = new HiruZenbaki();

		int ezkerrekoIndizea = erdia;
		int ezkerrekoBaturaMax=v[erdia];
		int ezkerrekoBaturaPartziala=v[erdia];
		for(int e=erdia-1; e>=hasiera; e--){
			ezkerrekoBaturaPartziala=ezkerrekoBaturaPartziala+v[e];
			if(ezkerrekoBaturaPartziala>ezkerrekoBaturaMax){
				ezkerrekoBaturaMax=ezkerrekoBaturaPartziala;
				ezkerrekoIndizea=e;
			}
		}

		int eskuinekoIndizea = erdia+1;
		int eskuinekoBaturaMax = v[erdia+1];
		int eskuinekoBaturaPartziala = v[erdia+1];
		for(int e=erdia+2; e<=amaiera; e++){
			eskuinekoBaturaPartziala=eskuinekoBaturaPartziala+v[e];
			if(eskuinekoBaturaPartziala>eskuinekoBaturaMax){
				eskuinekoBaturaMax=eskuinekoBaturaPartziala;
				eskuinekoIndizea=e;
			}
		}
		h.batuketa=(ezkerrekoBaturaMax+eskuinekoBaturaMax);
		h.hasiera=ezkerrekoIndizea;
		h.amaiera=eskuinekoIndizea;
		return h;
	}
}
```

- K.AREN HAUTAKETA

```
public class KSelekzioa {

	public int banaketa(int [] v, int hasiera, int amaiera){
		int i1=hasiera;
		int i2=amaiera;
		//v[hasiera] pibotea da
		while (i1<i2){
			while (v[i1]<=v[hasiera] && i1<i2) i1++;
			while (v[i2]>v[hasiera]) i2--;
			if (i1<i2){
				swap(v, i1, i2);
				i1++;
				i2--;
			}
		}
		swap(v, hasiera, i2);
		return i2;
	}

	public void swap (int [] v, int i, int j){
		try{
			int aux;
			aux=v[i];
			v[i]=v[j];
			v[j]=aux;
		}catch(Exception e){
			System.out.println("Array hutsa");
		}
	}

	public int kSelekzioa(int [] v, int hasiera, int amaiera, int k){
		// s = k-ren posizio erlatiboa bektorean
		int s = hasiera+k-1;
		if (amaiera-hasiera+1 == 1){
			return v[hasiera];
		}else{
			int b = banaketa(v,hasiera,amaiera);
			if (s==b) return v[s];
			else if (b<s) return kSelekzioa(v, b+1, amaiera, s-b);
			else return kSelekzioa(v, hasiera, b-1, k);
		}
	}

}
```

Beste banaketa algoritmo bat:

```
public int banaketa(int []v, int hasiera, int amaiera){
	//amaiera=v.length-1
	int pibotea=v[hasiera];
	int banaketaPuntua=hasiera;
	for(int i=hasiera+1; i<=amaiera;i++){
		if(v[i]<pibotea){
			banaketaPuntua++;
			swap(v,banaketaPuntua,i);
		}
	}
	swap(v, hasiera, banaketaPuntua);
	return banaketaPuntua;
}
```

- QUICKSORT

```
public class QuickSort {

	public int banaketa(int [] v, int hasiera, int amaiera){
		int i1=hasiera;
		int i2=amaiera;
		//v[hasiera] pibotea da
		while (i1<i2){
			while (v[i1]<=v[hasiera] && i1<i2) i1++;
			while (v[i2]>v[hasiera]) i2--;
			if (i1<i2){
				swap(v, i1, i2);
				i1++;
				i2--;
			}
		}
		swap(v, hasiera, i2);
		return i2;
	}

	public void swap (int [] v, int i, int j){
		try{
			int aux;
			aux=v[i];
			v[i]=v[j];
			v[j]=aux;
		}catch(Exception e){
			System.out.println("Array hutsa");
		}
	}

	public void quickSort(int [] v, int hasiera, int amaiera){
		if ((amaiera-hasiera+1)>1){
			int banaketaPuntua = banaketa(v, hasiera, amaiera);
			quickSort(v, hasiera, banaketaPuntua-1);
			quickSort(v, banaketaPuntua+1, amaiera);
		}
	}

}
```

- MERGESORT

```
public class MergeSort {

	public void mergeSort (int [] v, int hasiera, int amaiera){
		int erdia;
		if (hasiera < amaiera){
			erdia = (hasiera+amaiera)/2;
			mergeSort(v, hasiera, erdia);
			mergeSort(v, erdia+1, amaiera);
			merge(v, hasiera, erdia, amaiera);
		}
	}

	private void merge(int[] v, int hasiera, int erdia, int amaiera)
{
		int i1, i2, iaux;
		int [] vaux = new int[v.length];
		i1=hasiera;
		i2=erdia+1;
		iaux=hasiera;
		while ((i1<=erdia) && (i2<=amaiera)){
			if(v[i1]<=v[i2]){
				vaux[iaux]=v[i1];
				i1++;
			}else{
				vaux[iaux]=v[i2];
				i2++;
			}
			iaux++;
		}
		if (i1==erdia+1){
			//vaux[i3..amaiera]=v[i2..amaiera]
			kopiatu(v, i2, amaiera, vaux, iaux);
		}else{
			kopiatu(v, i1, erdia, vaux, iaux);
		}
		kopiatu(vaux, hasiera, amaiera, v, hasiera);
	}

	private void kopiatu(int[] jatorria, int hasieraInd, int
amaieraInd, int[] kopia, int kopiaInd) {
		for(int i=hasieraInd; i<=amaieraInd; i++){
			kopia[kopiaInd]=jatorria[i];
			kopiaInd++;
		}
	}

}
```

- HEAPSORT

```
// MAXIMOAREN PROPIETATEA BETETZEN DUEN META
public class Meta {

	public void metaEraikiHondoratuz(int[] v){
		for(int i=(v.length-1)/2; i>=0; i--){
			hondoratu(v,i, v.length-1);
		}
	}

	public void hondoratu(int [] v,int indizea, int tamaina){
		int i = indizea;
		int aux;
		boolean jarraitu=true;
		while (((2*i)+1)<=(tamaina) && jarraitu){
			int umeMax=(2*i)+1;
			if (((2*i)+2)<=(tamaina) && v[(2*i)+2]>v[(2*i)+1]){
				umeMax=(2*i)+2;
			}
			if(v[i]<v[umeMax]){
				aux=v[i];
				v[i]=v[umeMax];
				v[umeMax]=aux;
				i=umeMax;
			}else{
				jarraitu=false;
			}
		}
	}

	public void heapsort(int[] v){
		metaEraikiHondoratuz(v);
		int amaiera=v.length-1;
		for(int i=v.length-1; i>0; i--){
			swap(v,0,i);
			amaiera--;
			hondoratu(v, 0, amaiera);
		}
	}

	private void swap(int[] v, int i, int i2) {
		int aux;
		aux=v[i];
		v[i]=v[i2];
		v[i2]=aux;
	}

}
```

- GRAFOEN KORRITZEAK

- Orokorrean

```
public void korritu(){
	for (Erpina e: auzokideZerrenda){
		e.setAztertua(false);
	}
	for(Erpina e: auzokideZerrenda){
		if(!e.aztertua()){
			sk(e); /* edo */ zk(e);
		}
	}
}
```

- Sakoneran

```
public void sk(Erpina e){
	e.setAztertua(true);
	int [] auzokideak = e.getAuzokideak();
	for (int a=0; a<auzokideak.length;a++){
		Erpina auzokidea = auzokideZerrenda[auzokideak[a]]
		if(!auzokidea.aztertua()){
			sk(auzokidea);
		}
	}
}
```

- Zabaleran

```
public void zk(Erpina e){
	e.setAztertua(true);
	ArrayDeque<Erpina> ilara = new ArrayDeque<Erpina>();
	ilara.add(e);
	while (!ilara.isEmpty()){
		Erpina aux = ilara.removeFirst();
		int [] auzokideak = aux.getAuzokideak();
		for(int i=0; i<auzokideak.length; i++){
			Erpina auzokidea = auzokideZerrenda[auzokideak[i]];
			if(!auzokidea.aztertua()){
				auzokidea.setAztertua(true);
				ilara.add(auzokidea);
			}
		}
	}
}
```

- KOSTUAK

- **Ordenazio algoritmoak** *[n=bektoreko elementu kopurua].*

ALGORITMOA	ORDENA [Kasu txarrena]
Txertaketa	$O(n^2)$
Burbuila	$O(n^2)$
Quicksort	$O(n^2)$ Batezbesteko:O(n lgn)
MergeSort	O(n lgn)
HeapSort	O(n lgn)

- **Metak** *[n=metako elementu kopurua]*

PROZEDURA	ORDENA [Kasu txarrena]
Hondoratu	O(lgn)
Azaleratu	O(lgn)
MetaEraikiHondoratuz	O(n)
MetaEraikiAzaleratuz	O(n lgn)

- **Bilaketa dikotomikoa**: O(lgn) *[n=bektoreko elementu kopurua].*

- **K. hautaketa**: Kasu txarrenean $O(n^2)$ eta batezbestekoan O(n). Algoritmoaren barnean erabiltzen den **Banaketa** prozedurak O(n)-ko kostua du (Quicksorten ere erabiltzen da). *[n=bektoreko elementu kopurua].*

- **Batura maximoko segmentua**: O(n lgn) *[n=bektoreko elementu kopurua].*

- **Grafoen korritzeak** (sakonerakoa eta zabalerakoa): O(n+a) *[n=erpin kopurua eta a=arku/ertz kopurua].*